



Android forensics deep dive

Acquisition & analysis of Raw NAND flash and the YAFFS2 file system

Dr. Bradley Schatz

Director, Schatz Forensic

Adjunct associate professor, Queensland
University of Technology

AusCERT 2013 – Gold Coast



Background



NAND Flash is used pervasively in mobile and embedded devices

- Mobile phones retain:
 - What they said,
 - Who they said it to,
 - Where they said it,
 - What they searched for (what concerns them)
 - Where they travelled
 - When they charged
 - ...



NAND Flash is used pervasively in mobile and embedded devices

- Mobile phones retain:
 - What they said,
 - Who they said it to,
 - Where they said it,
 - What they searched for (what concerns them)
 - Where they travelled
 - When they charged
 - ...



Significant challenges to establishing reliable evidence

- Getting the data out intact
 - Device and OS diversity
- Interpreting the data into usable evidence
 - Device and OS diversity
- Absence of scientific rigour from tool vendors
 - Transparency & independent reproducibility missing to date

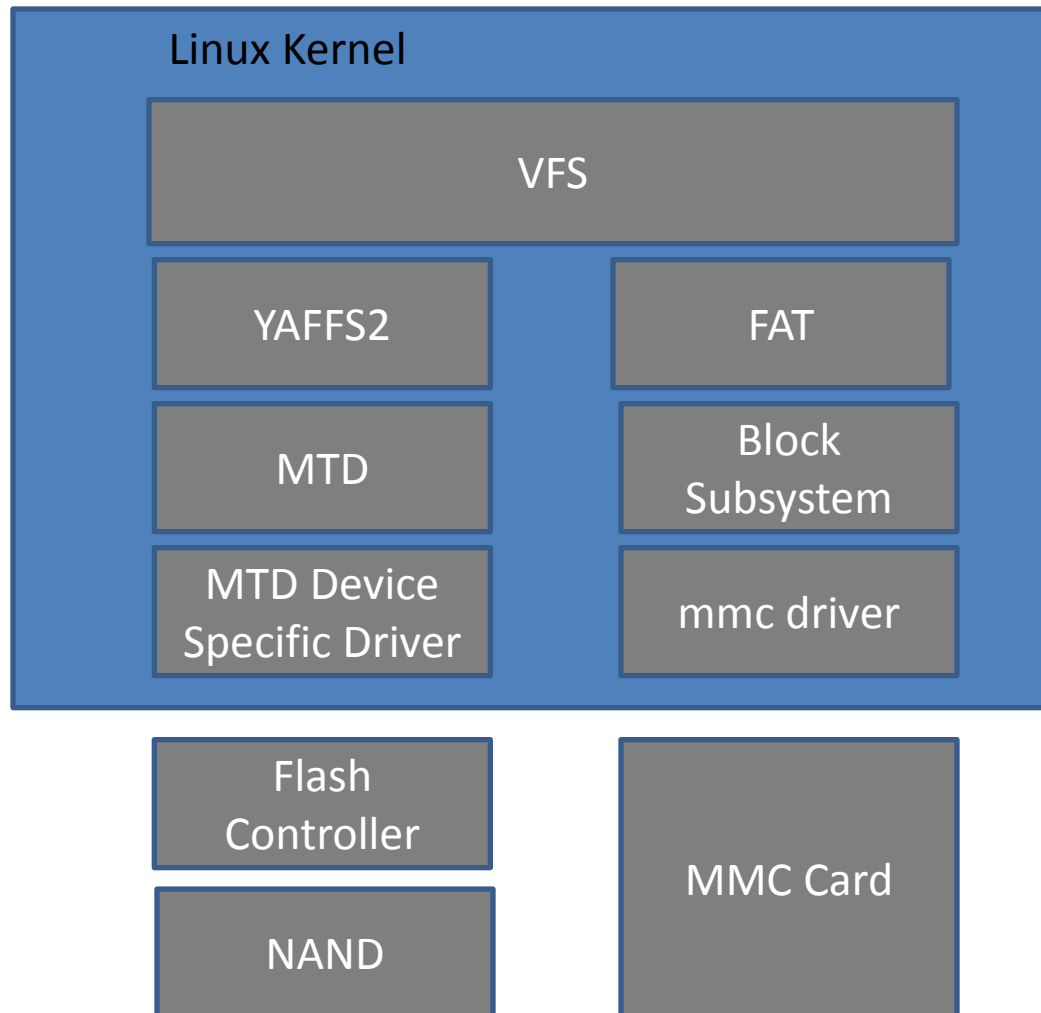
The Android landscape isn't homogenised

- Bootloader: Redboot, HTC HBoot, Samsung
- Filesystem: **YAFFS2**, Samsung RFS, EXT4
- FTL: Integrated, MTD, Samsung XSR
- Memory device: **Raw NAND flash** (xN footprint), eMMC...

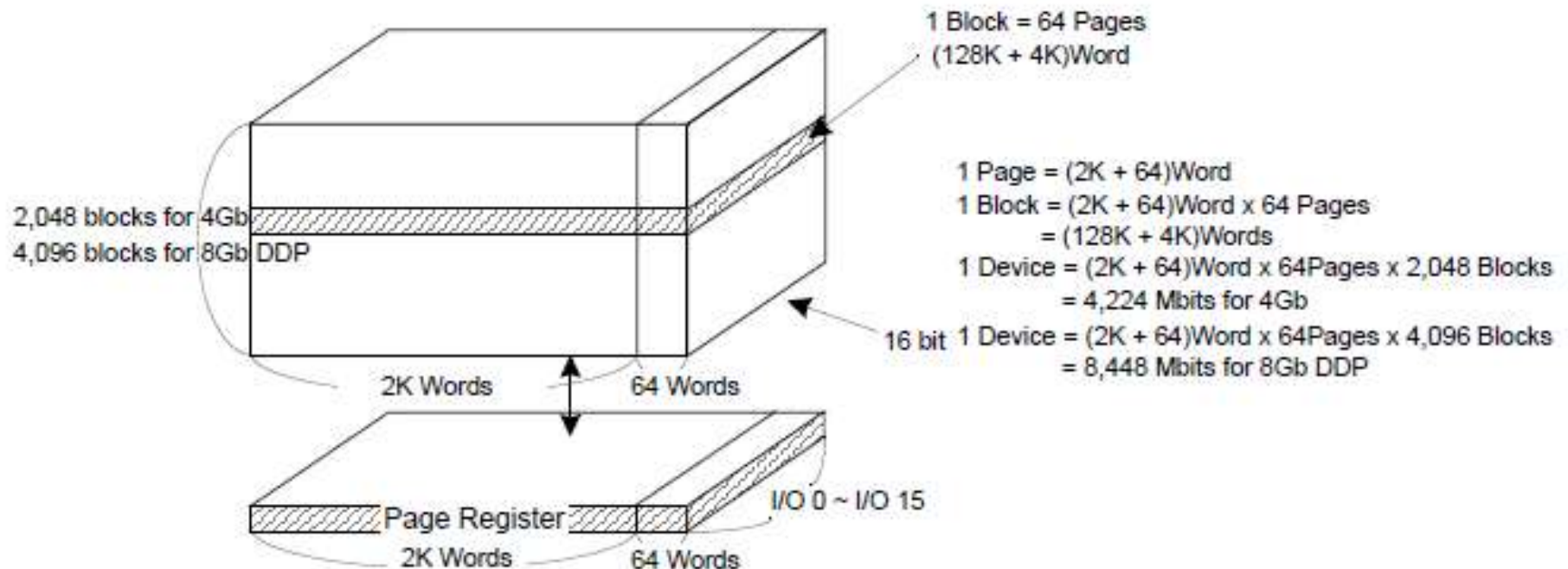


Theory of operation

An Android Storage Architecture

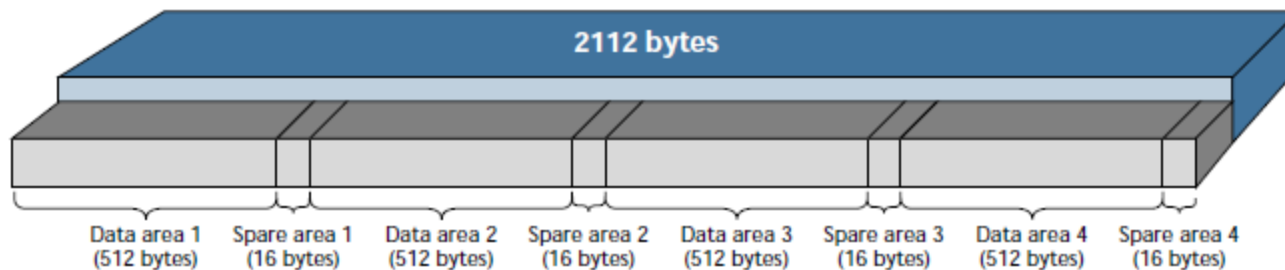


Flash memory is designed to store metadata in addition to each block

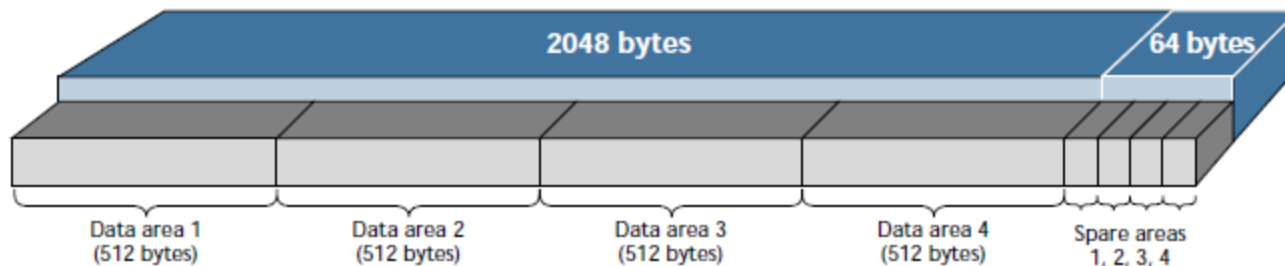


The metadata and data may be arranged differently in a page

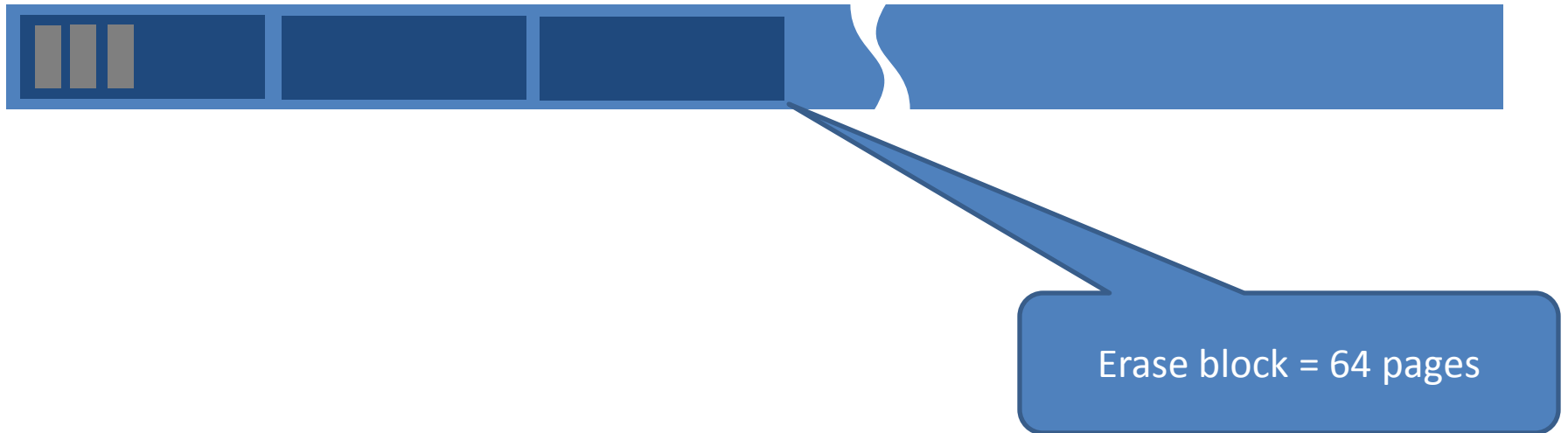
Adjacent Data and Spare Areas



Separate Data and Spare Areas



Pages cannot be individually erased



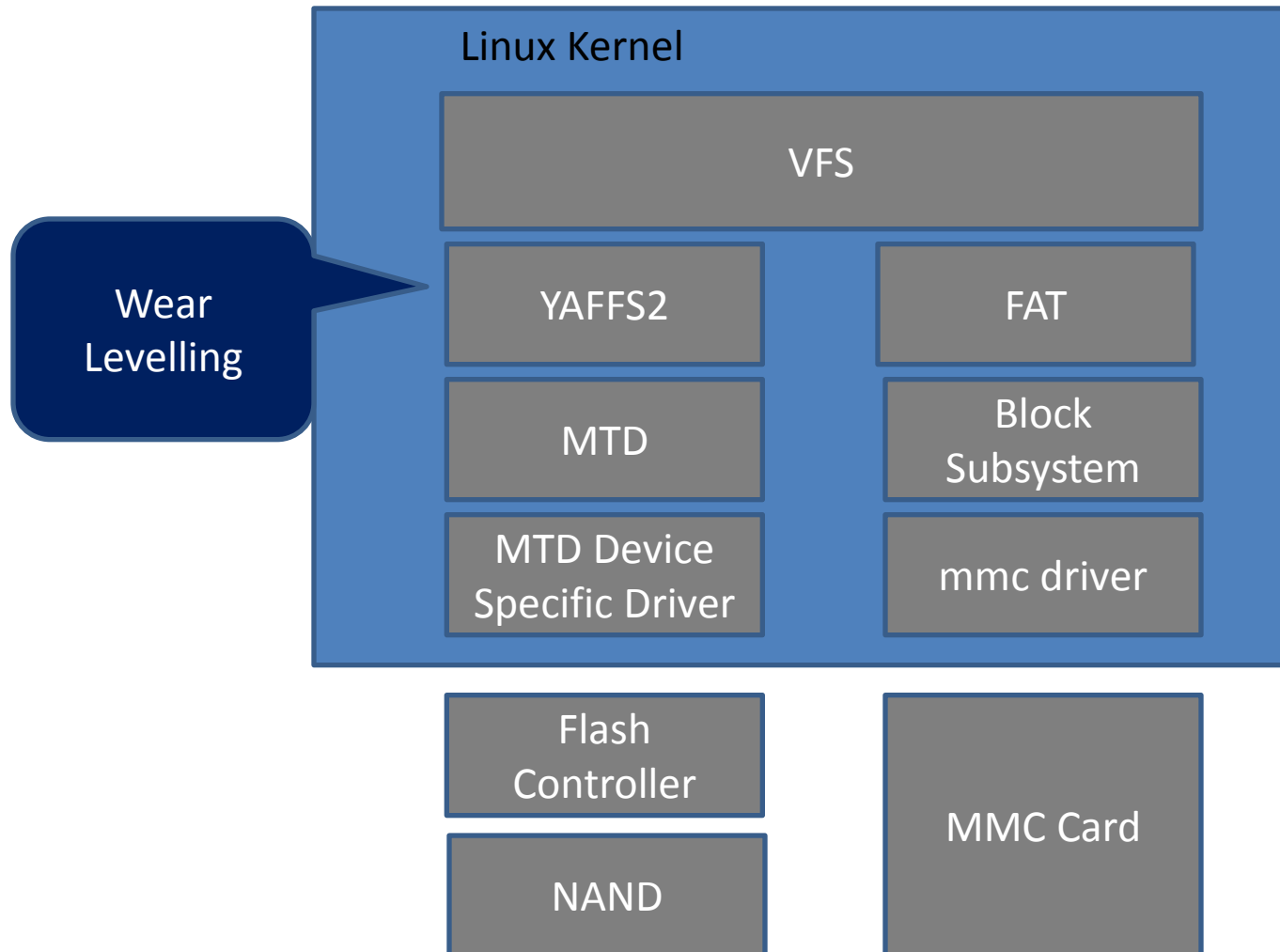
Pages must be written serially to an Erase Block



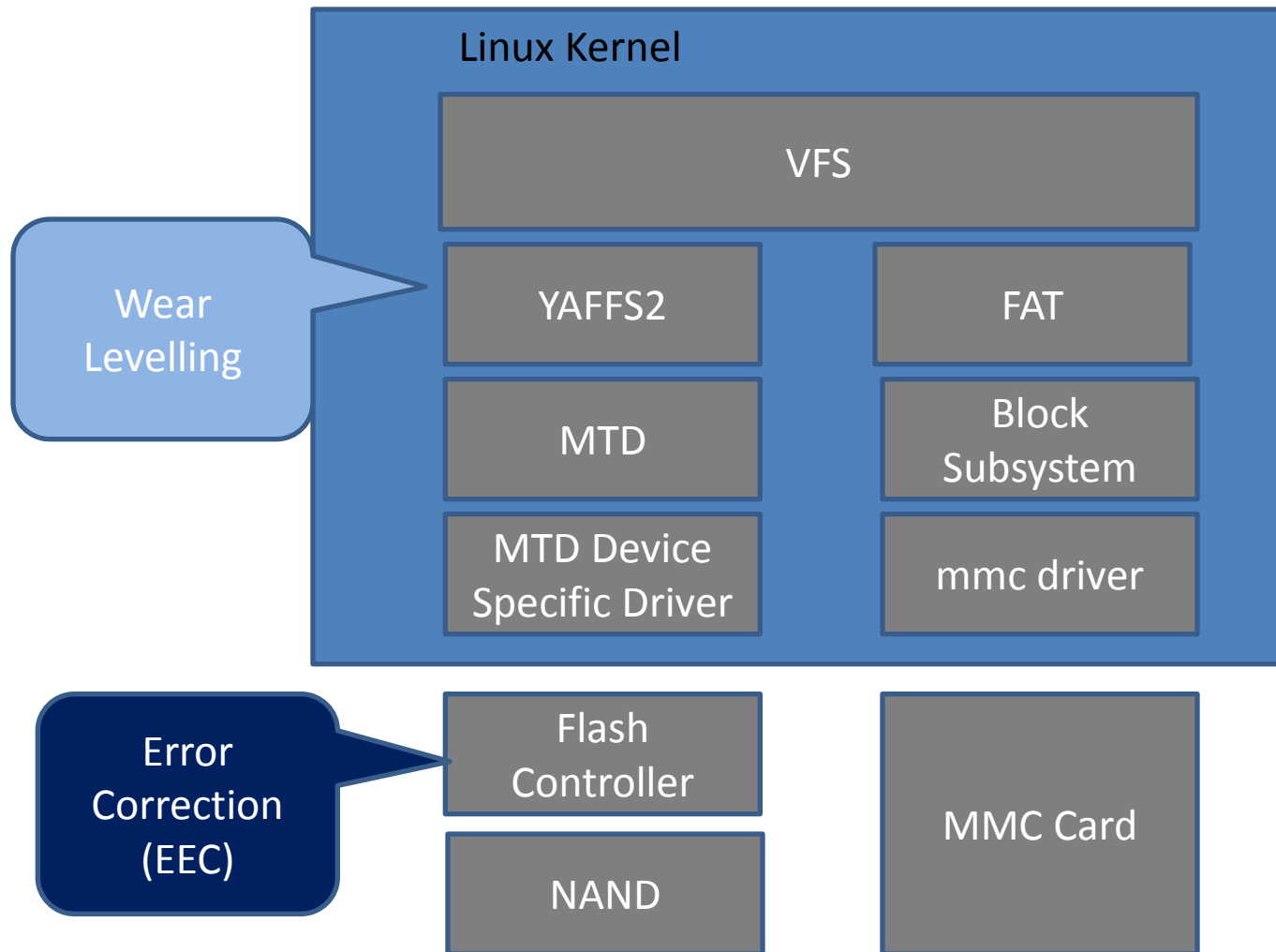
Page written before those to left



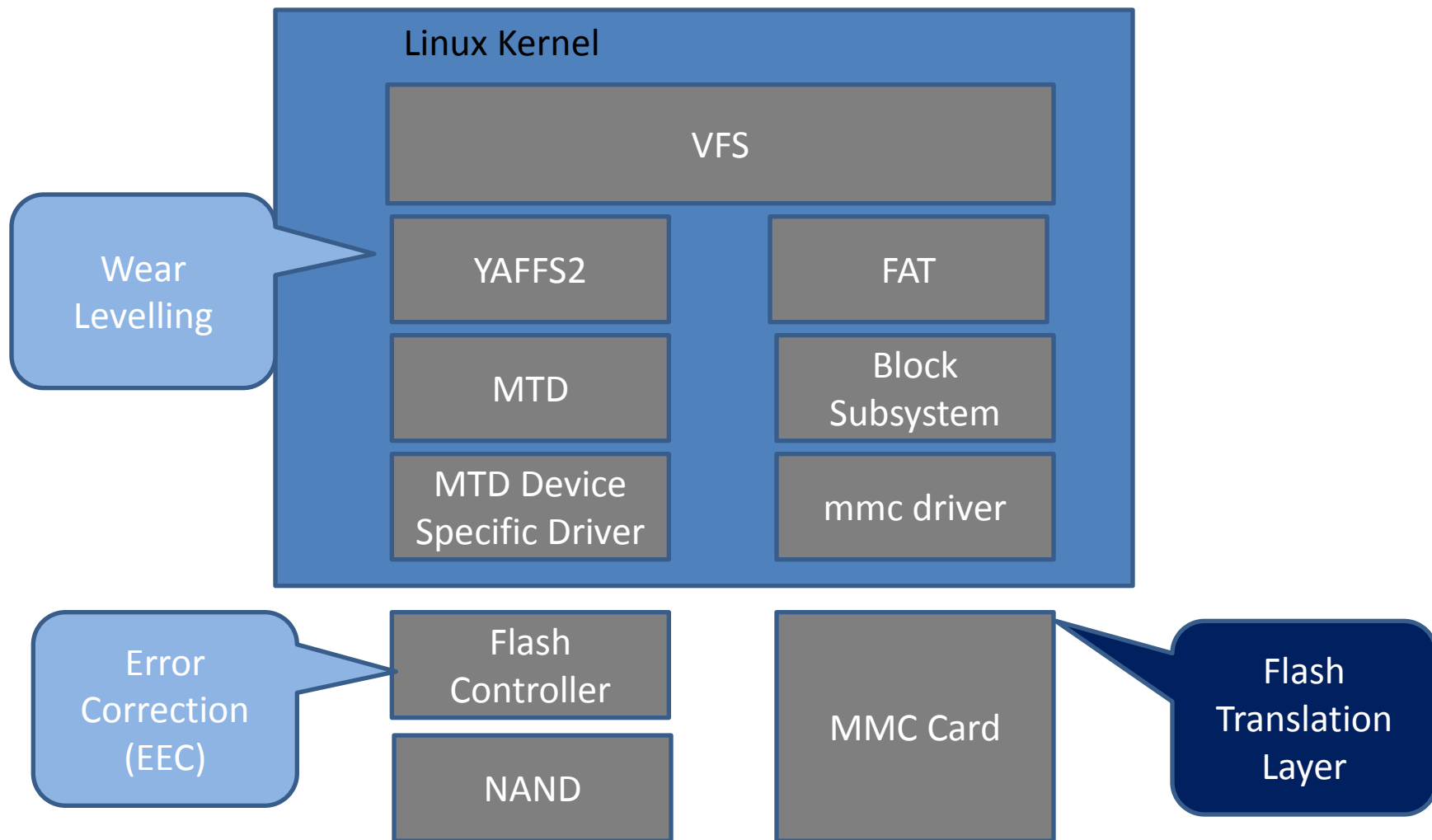
Pages can only be written a fixed number of times



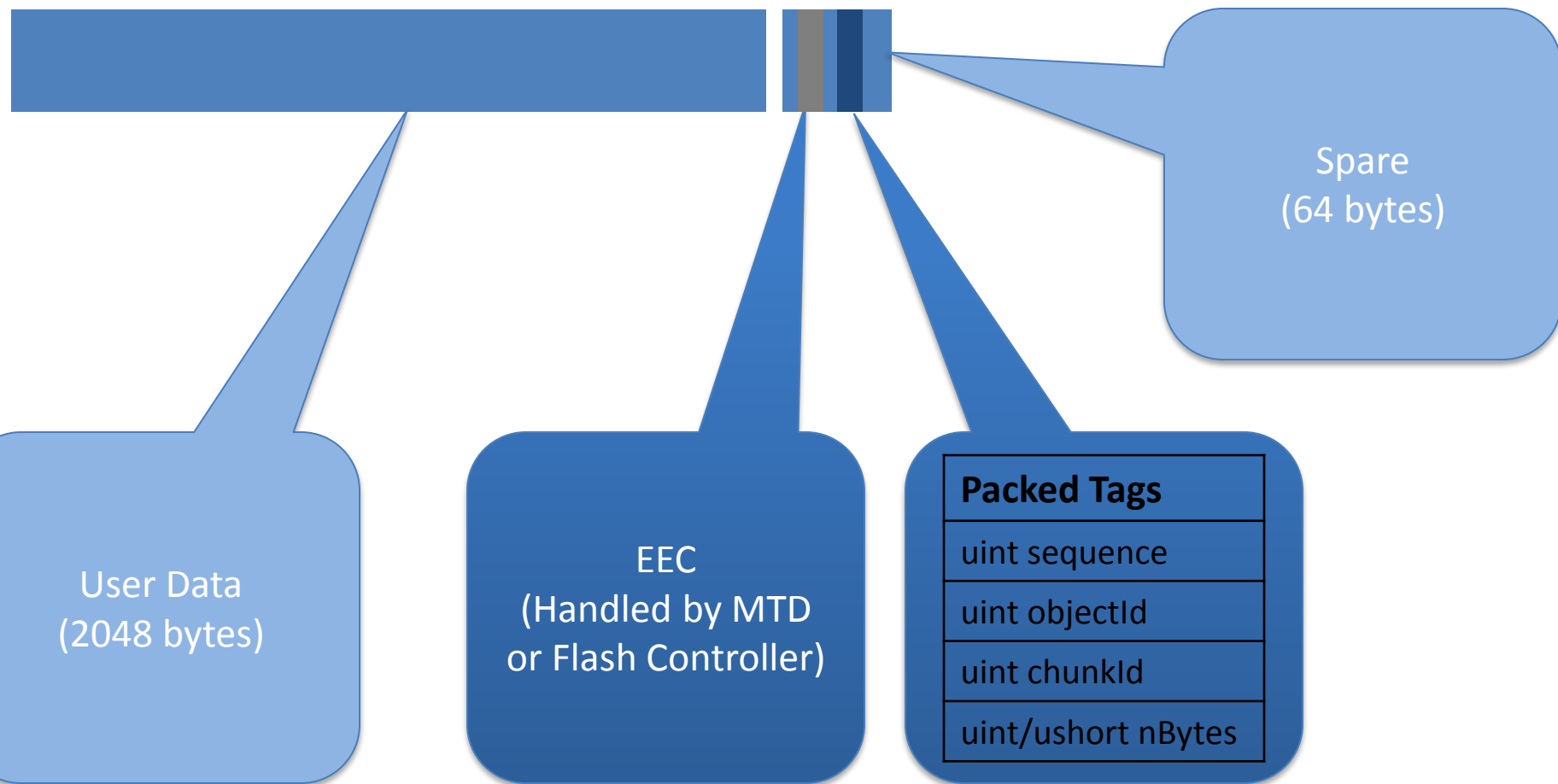
Bit errors are anticipated



MMC Integrates flash controller and NAND



YAFFS2 stores metadata in the spare



YAFFS2 Basics: Simple file write



Object Header

ChunkID = 0

File Name = "a"

Sequence = 0x1001

Size = 0

Data block 1

ChunkID = 1

Address = 0x0

Sequence = 0x1001

Data block 2

ChunkID = 2

Address = 2048

Sequence = 0x1001

Object Header

ChunkID = 0

File Name = "a"

Sequence = 0x1001

Size = 4096



Getting the data out

Acquisition



Acquisition principles

- Completeness
- Accuracy
- Repeatability
- Integrity



Acquisition methods

- Logical: file copy over android debug bridge
- Pseudo physical: get root, dump NAND block devices
- Bootloader
- Physical 1: JTAG access to flash
- Physical 2: Chip off



Logical

- Enable ADB on phone
- Connect and recursive copy
- - Limited access to files
- - No prior versions
- - We are trusting the kernel

Bootloader approaches

- Disable bootloader security
- RAM load custom boot image
- Dump using “Live” Ramdisk
- - Wipes most (not all) devices
- - Limited coverage
- - Maintenance of “live” ramdisks

See: Cannon (2012) Into the Driod, Blackhat

See: Vidas (2011) Toward a general collection methodology for Android devices, DFRWS

Pseudo-physical (nanddump)

- Requires unlocked phone/pin
- Requires root access to device
 - Range of exploits to do this
 - - Exploit validation
 - - Perception management
- Dump MTD devices with nanddump
 - - MTD is not accurate
 - - We still don't have access to the entire flash device
 - - We are trusting the phone's kernel

Pseudo-physical: Exploitation

```
<3>[ 684.803710] init: untracked pid 3882 exited
<3>[ 684.803833] init: untracked pid 3883 exited
<3>[ 684.803955] init: untracked pid 3884 exited
<3>[ 684.804199] init: untracked pid 3885 exited
<3>[ 684.804321] init: untracked pid 630 exited
<6>[ 723.455749] [HTC_BATT]RSNSP=67,RARC=6,Vol=3781mV,Current=299mA,Temp=288C(1/10)
<6>[ 723.455963] batt: ds2784_notify: 1 6 at 719276978798 (1980-01-06 00:14:06.669006333 UTC)
<6>[ 723.462738] batt: batt:power_supply_changed: battery at 719283875771 (1980-01-06 00:14:06.675750718 UTC)
<6>[ 783.453399] [HTC_BATT]RSNSP=67,RARC=6,Vol=3781mV,Current=301mA,Temp=288C(1/10)
<6>[ 843.457244] [HTC_BATT]RSNSP=67,RARC=7,Vol=3781mV,Current=301mA,Temp=288C(1/10)
<6>[ 843.457458] batt: ds2784_notify: 1 7 at 839278474159 (1980-01-06 00:16:06.670501694 UTC)
<6>[ 843.464019] batt: batt:power_supply_changed: battery at 839285035439 (1980-01-06 00:16:06.677062974 UTC)
<6>[ 903.451873] [HTC_BATT]RSNSP=67,RARC=7,Vol=3781mV,Current=301mA,Temp=290C(1/10)

# cat /proc/mtd
dev:   size   erasesize  name
mtd0: 000a0000 00020000 "misc"
mtd1: 00500000 00020000 "recovery"
mtd2: 00280000 00020000 "boot"
mtd3: 0fa00000 00020000 "system"
mtd4: 02800000 00020000 "cache"
mtd5: 093a0000 00020000 "userdata"
```

Pseudo-physical: Get I/O Channel

```
# cat /      mount

rootfs / rootfs ro 0 0

tmpfs /dev tmpfs rw,mode=755 0 0

devpts /dev/pts devpts rw,mode=600 0 0

proc /proc proc rw 0 0

sysfs /sys sysfs rw 0 0

tmpfs /sqlite_stmt_journals tmpfs rw,size=4096k 0 0

none /dev/cpuctl cgroup rw,cpu 0 0

/dev/block/mtdblock3 /system yaffs2 ro 0 0

/dev/block/mtdblock5 /data yaffs2 rw,nosuid,nodev 0 0

/dev/block/mtdblock4 /cache yaffs2 rw,nosuid,nodev 0 0

tmpfs /app-cache tmpfs rw,size=8192k 0 0

/dev/block//vold/179:1 /sdcard vfat
rw,dirsync,nosuid,nodev,noexec,uid=1000,gid=1015,fmask=0702,dmask=0702,allow_utime=0020,codepage=cp437,iocharset=iso8859-
1,shortname=mixed,utf8,errors=remount-ro 0 0

# mount -o exec,remount /dev/block//vold/179:1 /sdcard

# cd /s sdcaard  rd
# chmod 755 nanddump

# ls -l

d---rwxr-x system  sdcard_rw          1980-01-06 10:01 LOST.DIR

----r-xr-x system  sdcard_rw  713750 2011-12-16 20:47 nanddump
```

Pseudo-physical: Acquire

```
# ls /dev -l /dev/md td/

crw----- root    root    90, 11 1980-01-06 10:02 mtd5ro
crw----- root    root    90, 10 1980-01-06 10:02 mtd5
crw----- root    root    90,  9 1980-01-06 10:02 mtd4ro
crw----- root    root    90,  8 1980-01-06 10:02 mtd4
crw----- root    root    90,  7 1980-01-06 10:02 mtd3ro
crw----- root    root    90,  6 1980-01-06 10:02 mtd3
crw----- root    root    90,  5 1980-01-06 10:02 mtd2ro
crw----- root    root    90,  4 1980-01-06 10:02 mtd2
crw----- root    root    90,  3 1980-01-06 10:02 mtd1ro
crw----- root    root    90,  2 1980-01-06 10:02 mtd1
crw----- root    root    90,  1 1980-01-06 10:02 mtd0ro
cr--rw---- radio   diag    90,  0 1980-01-06 10:02 mtd0

# ./nanddump --bb=dumpbad -o -f ./mtd0.nanddump /dev/mtd/mtd0

ECC failed: 0
ECC corrected: 0
Number of bad blocks: 0
Number of bbt blocks: 0
Block size 131072, page size 2048, oob size 56

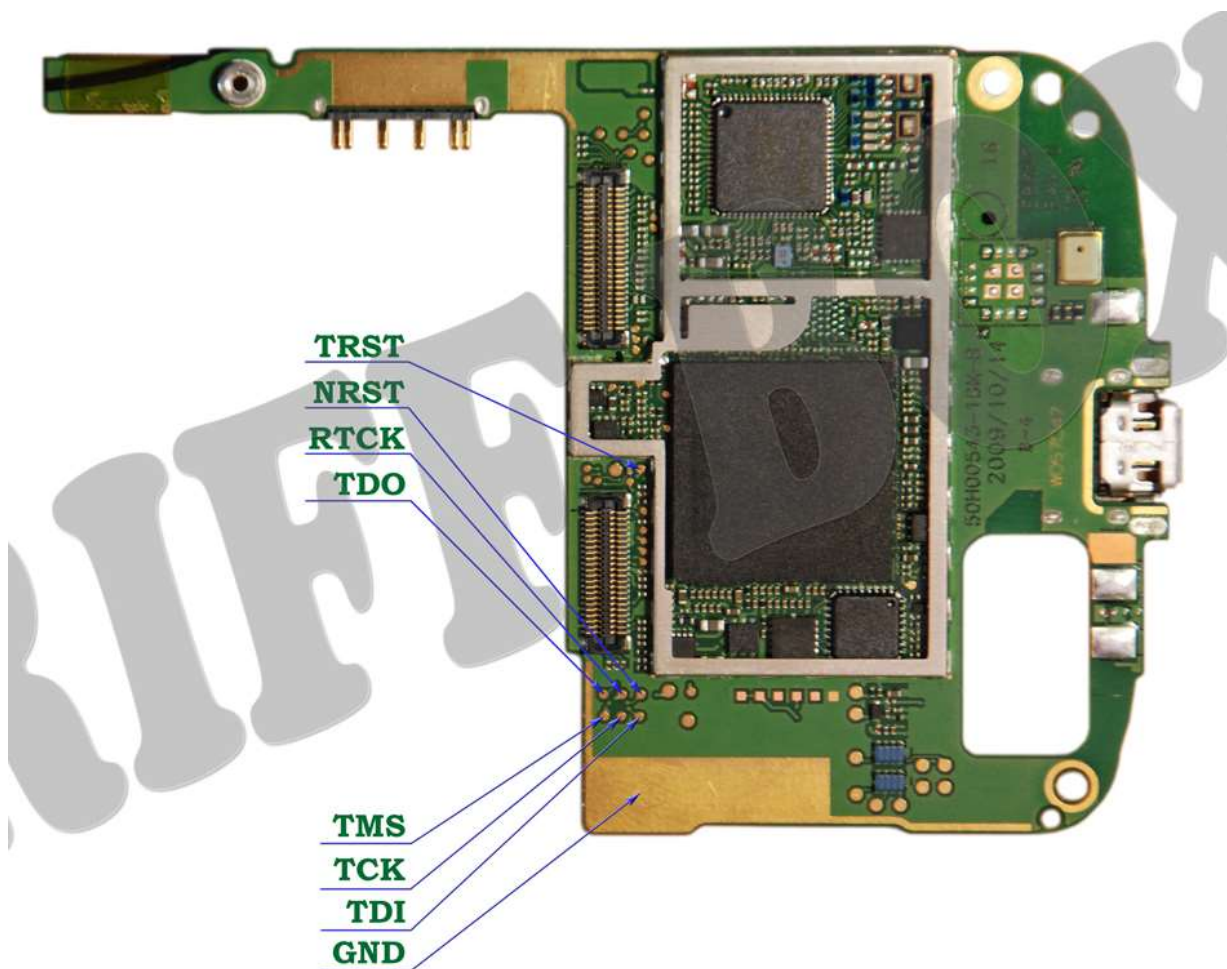
Dumping data starting at 0x00000000 and ending at 0x000a0000...
```

This doesn't
match our
theory of
operation

JTAG Acquisition: Dismantle phone

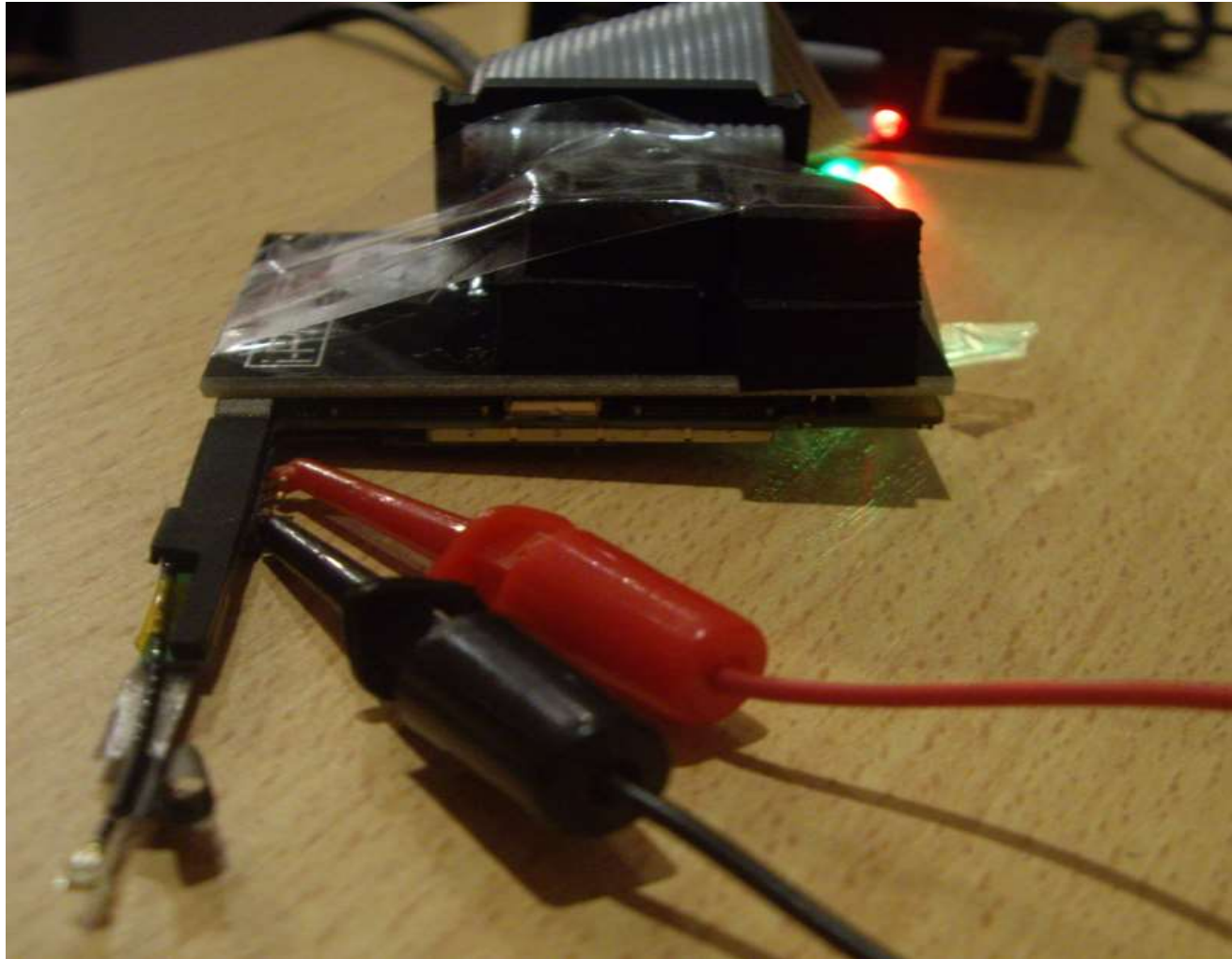


JTAG acquisition: Identify JTAG points



Source: RIFF Box JTAG Manager

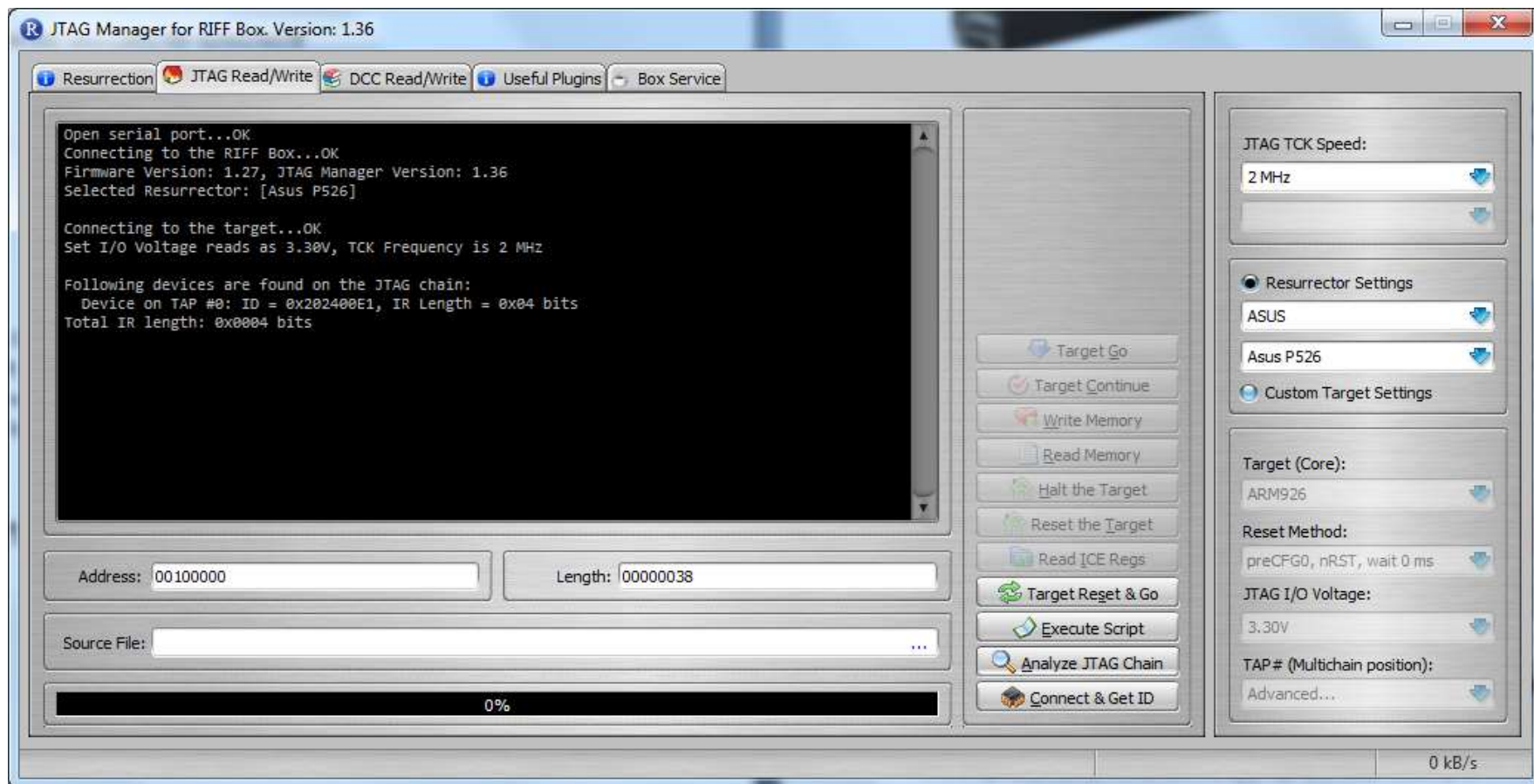
JTAG acquisition: Connect Jig & Power cables



JTAG acquisition: Connect to JTAG adapter



JTAG acquisition: Dump flash





JTAG acquisition

- ! Grey market hardware/software
- ! Finicky
- + Complete acquisition
- + No kernel involvement

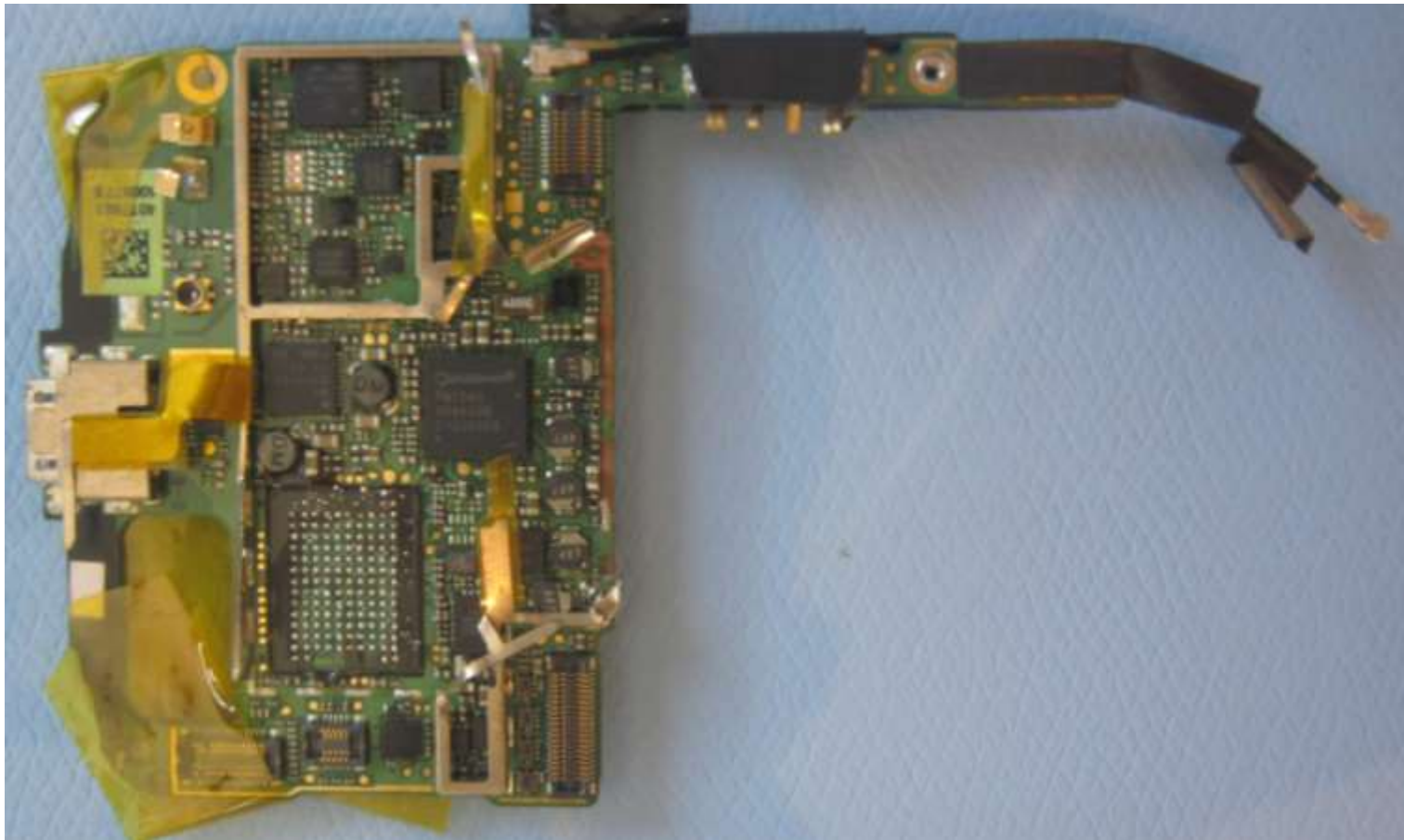


Chip off acquisition

- Dismantle phone++
- Identify flash
- Determine solder melting point
 - Lead free testing kit
- Remove flash
 - Kapton tape thermocouples to monitor temperature
 - Controlled heat to chip (BGA IR Rework or Hot Air)

137-Ball Fine pitch Ball Grid Array Package (measured in millimeters)

Chip off acquisition: post flash chip removal

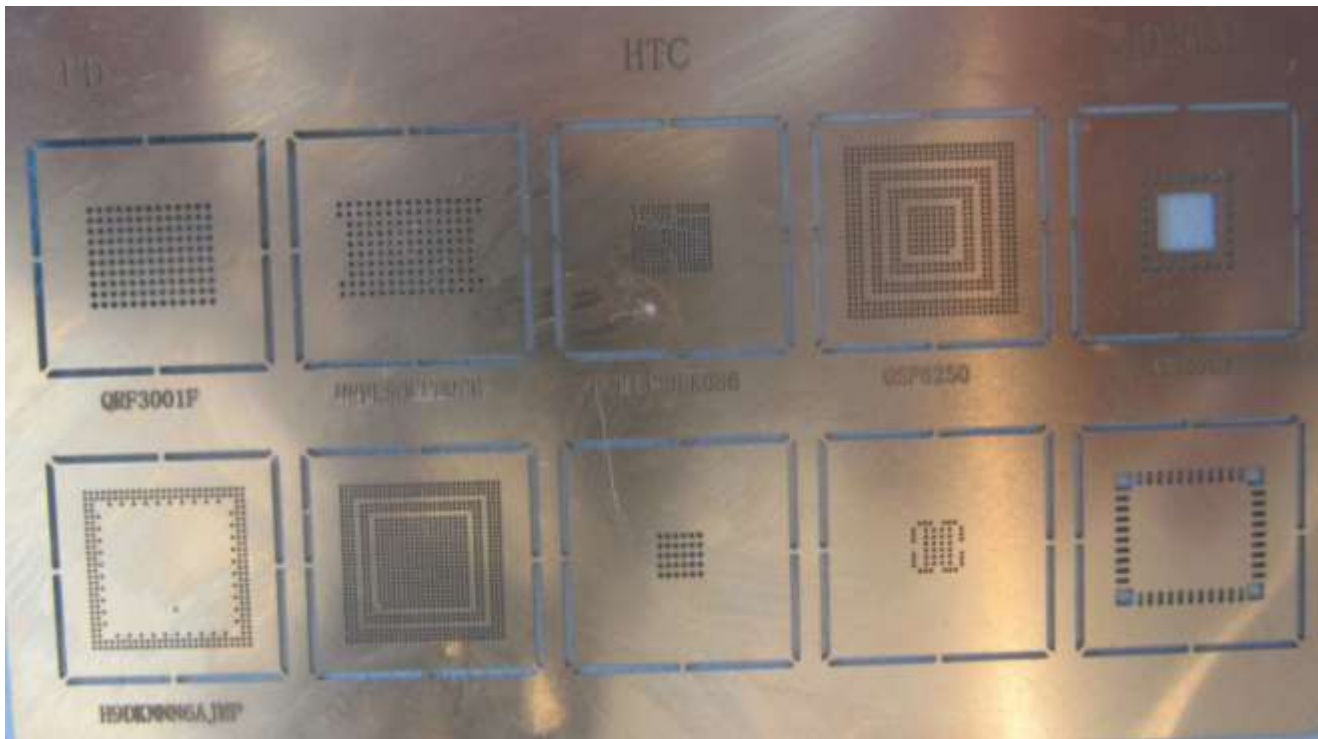


A microscopic view of a chip, showing a grid-like pattern of small, dark, rectangular components.

Chip off acquisition

- Cleaning of chip
 - Removal of excess solder
 - Removal of BGA underfill
 - Clean with Isopropyl alcohol
- Re-balling
 - Kapton tape chip to underside of stencil
 - Apply solder paste and squeegee
 - Melt solder with hot air

Chip off acquisition: re-balling stencil and re-balled chip





Chip off acquisition

- Acquire chip footprint specific adapter
 - Wide variety in chip sizes
- Acquire chip contents
 - Universal programmer
 - Build your own



Chip off acquisition

- ? Heat effects on flash content
- ? Moisture + heat effects
- ! Finicky++
- ! Expensive tools



Analysis of Flash Volume



Interpretation methodology

1. Determine flash image format
2. Identify partition layout
3. Yaffs2: Identify tags layout
 - Byte plots [1] as a perception enhancing tool
4. Interpret YAFFS Structures

[1] Conti et al (2010) Automated mapping of large binary objects using primitive fragment type classification

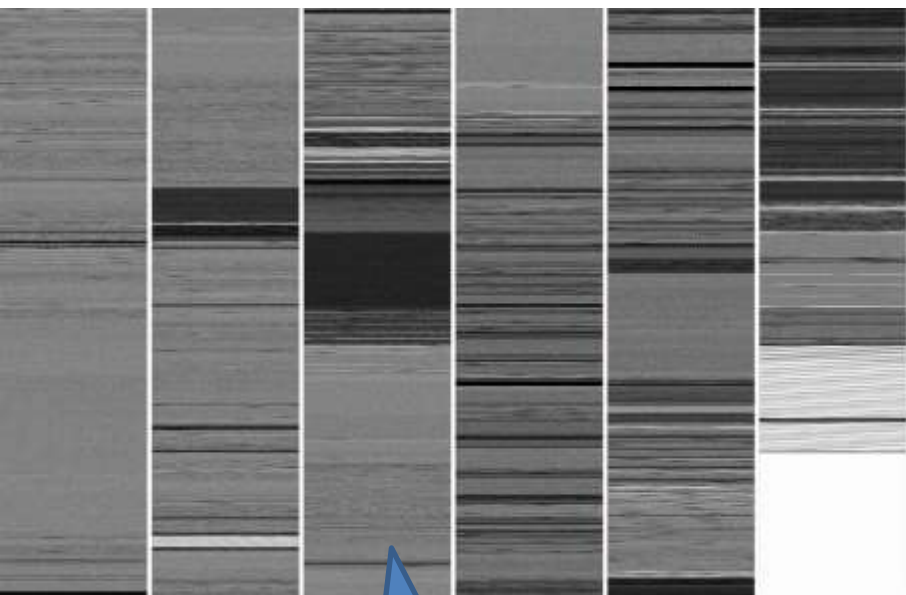


Byteplot tool

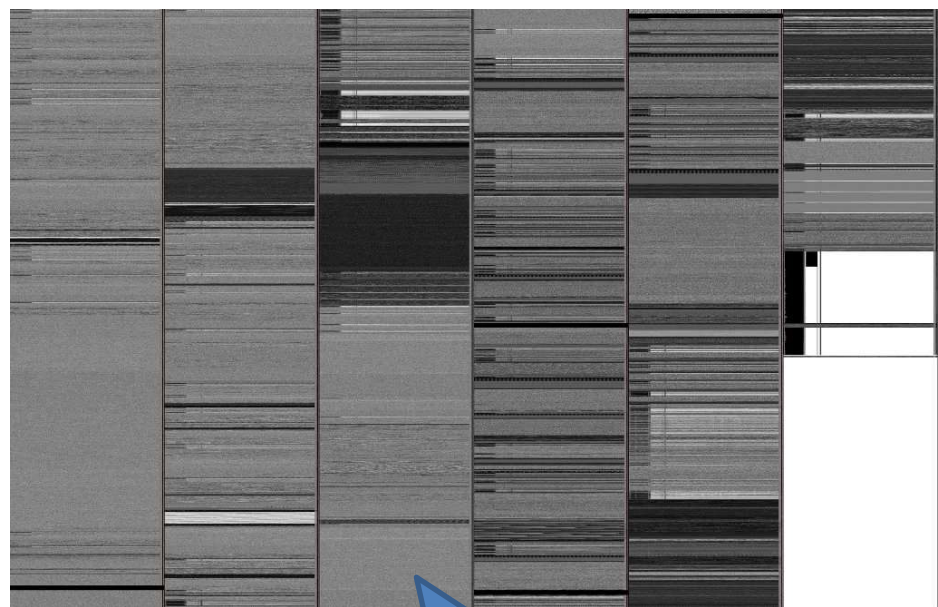
- Each byte in source image = one greyscale value [1]
- Organised with:
 - visual cues separating spare from user data area
 - multiple columns (populate down then right)

[1] Conti et al (2010) Automated mapping of large binary objects using primitive fragment type classification

Determine flash image format: inline spare vs End spare

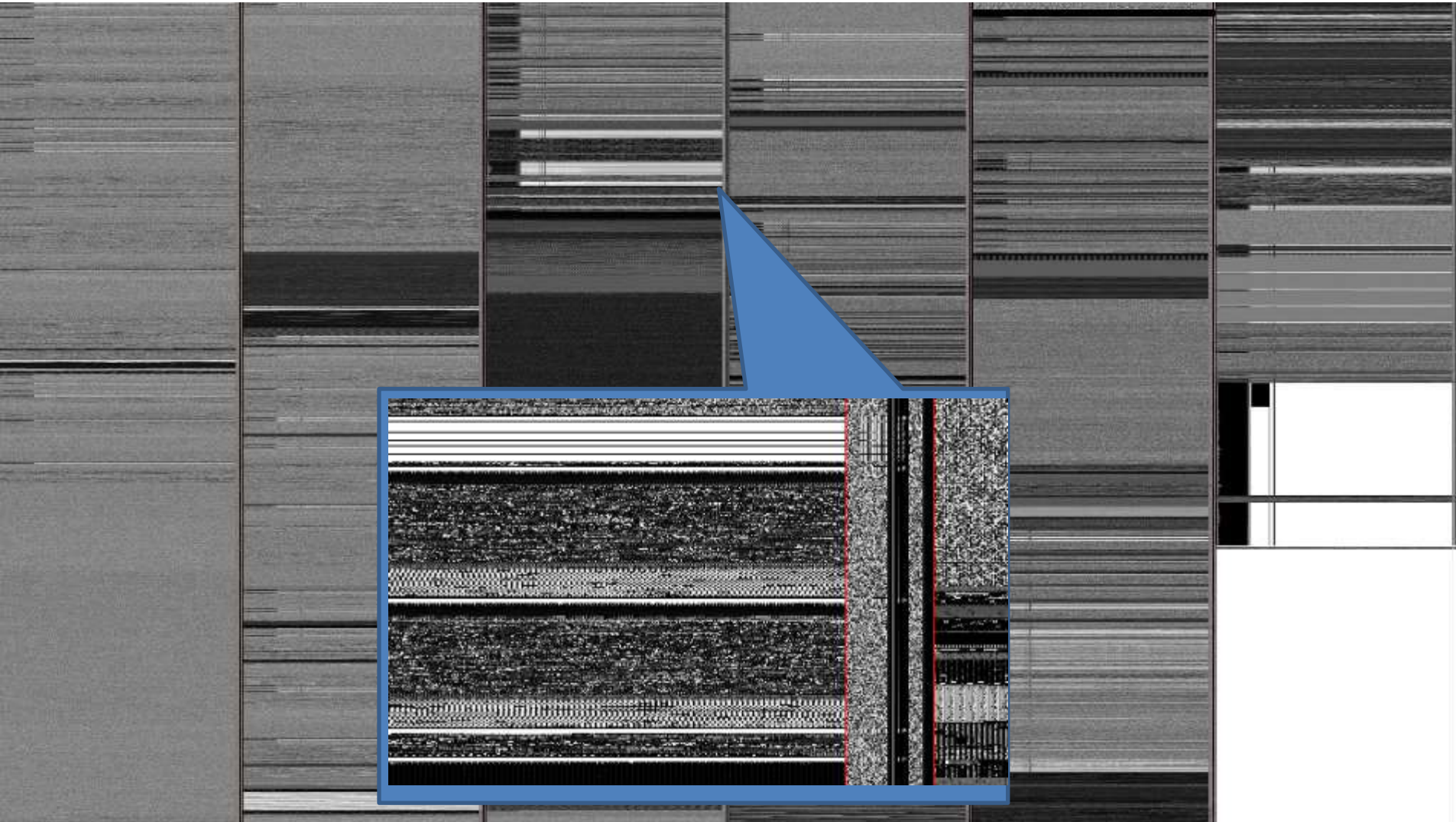


JTAG
acquisition w/
spare at end

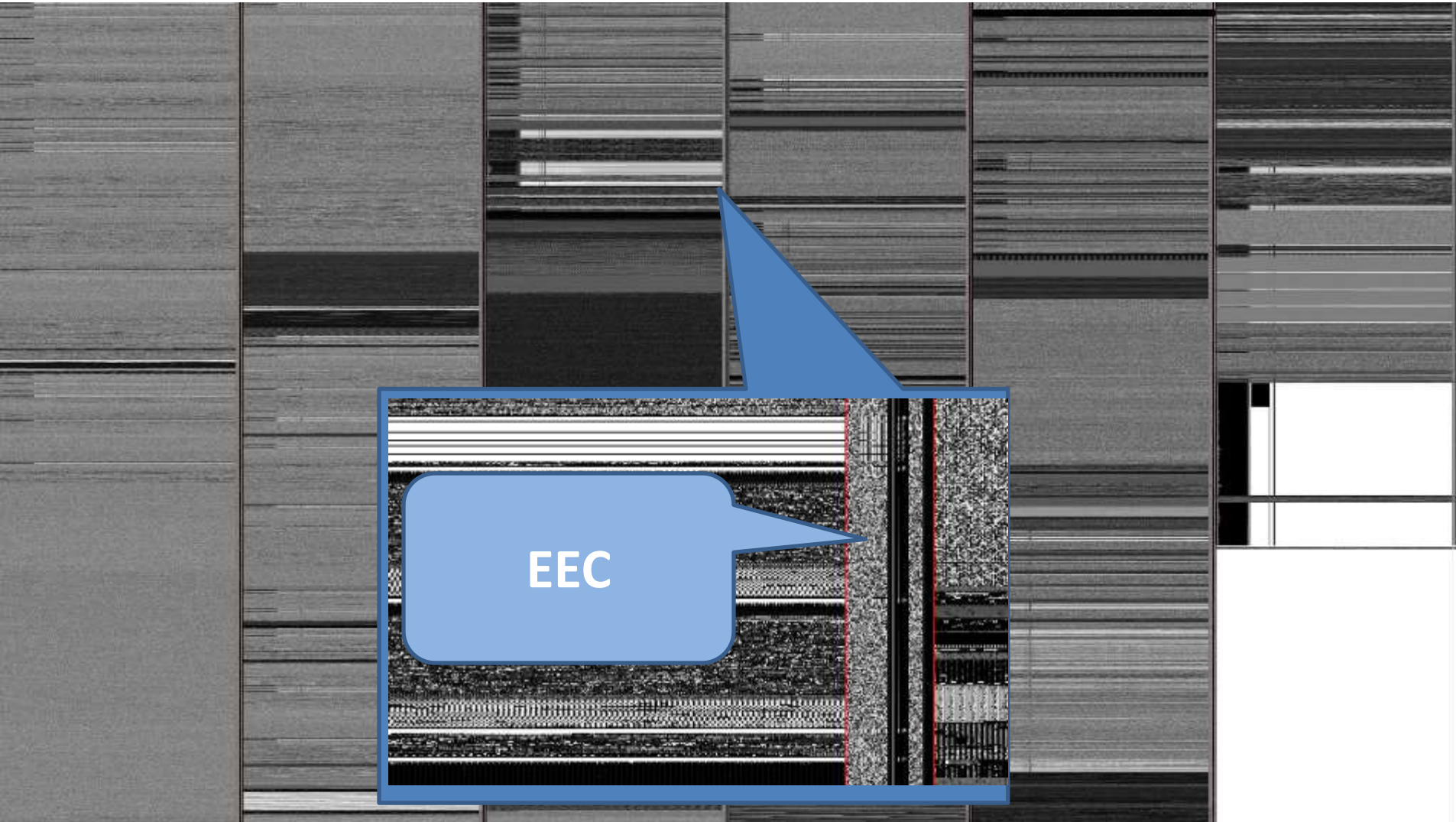


JTAG image normalised
to inline spare

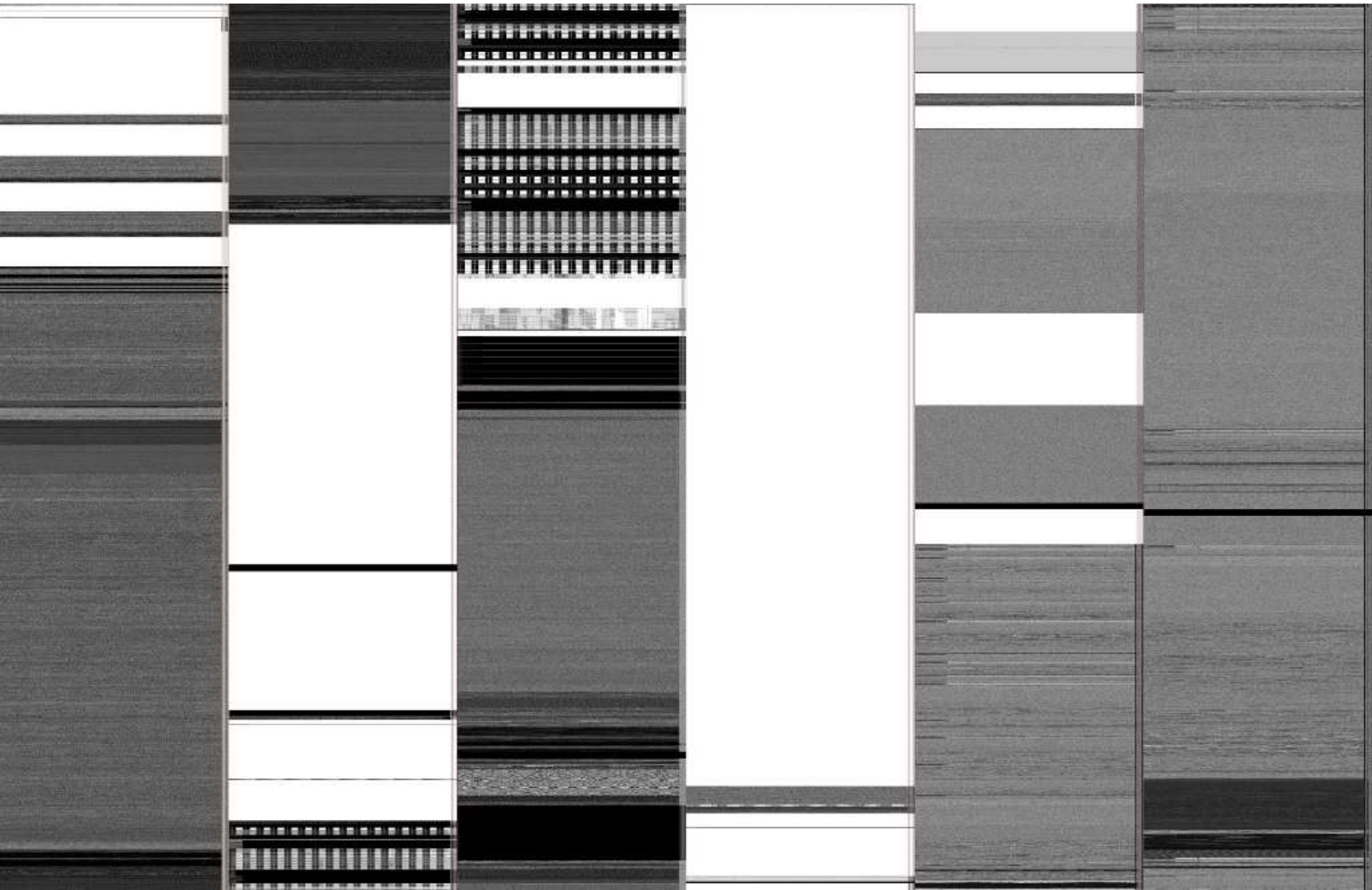
Clear delineation between spare and
user data



Clear delineation between spare and user data



Identification of the partition layout



Kernel dmesg flash partitioning is the most straightforward

```
<6>[ 10.202087] msm_nand: allocated dma buffer at ffa01000, dma_addr
3b1ac000
<6>[ 10.208343] msm_nand: read CFG0 = aa5400c0 CFG1 = 6746e
<6>[ 10.213317] msm_nand: CFG0 cw/page=3 ud_sz=512 ecc_sz=10 spare_sz=4
<6>[ 10.219757] msm_nand: NAND_READ_ID = 5500bcec
<6>[ 10.224060] msn_nand: nandid 5500bcec status c03120
<6>[ 10.228881] msm_nand: manuf Samsung (0xec) device 0xbc blocksz 20000
pagesz 800 size 20000000
<6>[ 10.237274] msm_nand: save CFG0 = e85408c0 CFG1 = 4745e
<6>[ 10.242584] msm_nand: CFG0: cw/page=3 ud_sz=516 ecc_sz=10 spare_sz=0
num_addr_cycles=5
<6>[ 10.250457] msm_nand: DEV_CMD1: f00f3000
<6>[ 10.254455] msm_nand: NAND_EBI2_ECC_BUF_CFG: 1ff
<5>[ 10.258911] Creating 6 MTD partitions on "msm_nand":
<5>[ 10.263946] 0x00001ff60000-0x000020000000 : "misc"
<5>[ 10.270080] 0x000004240000-0x000004740000 : "recovery"
<5>[ 10.279846] 0x000004740000-0x0000049c0000 : "boot"
<5>[ 10.283508] 0x0000049c0000-0x0000143c0000 : "system"
<5>[ 10.556365] 0x0000143c0000-0x000016bc0000 : "cache"
<5>[ 10.600402] 0x000016bc0000-0x00001ff60000 : "userdata"
```

/proc/mtd doesn't give offset (and the partitions may be out of order)

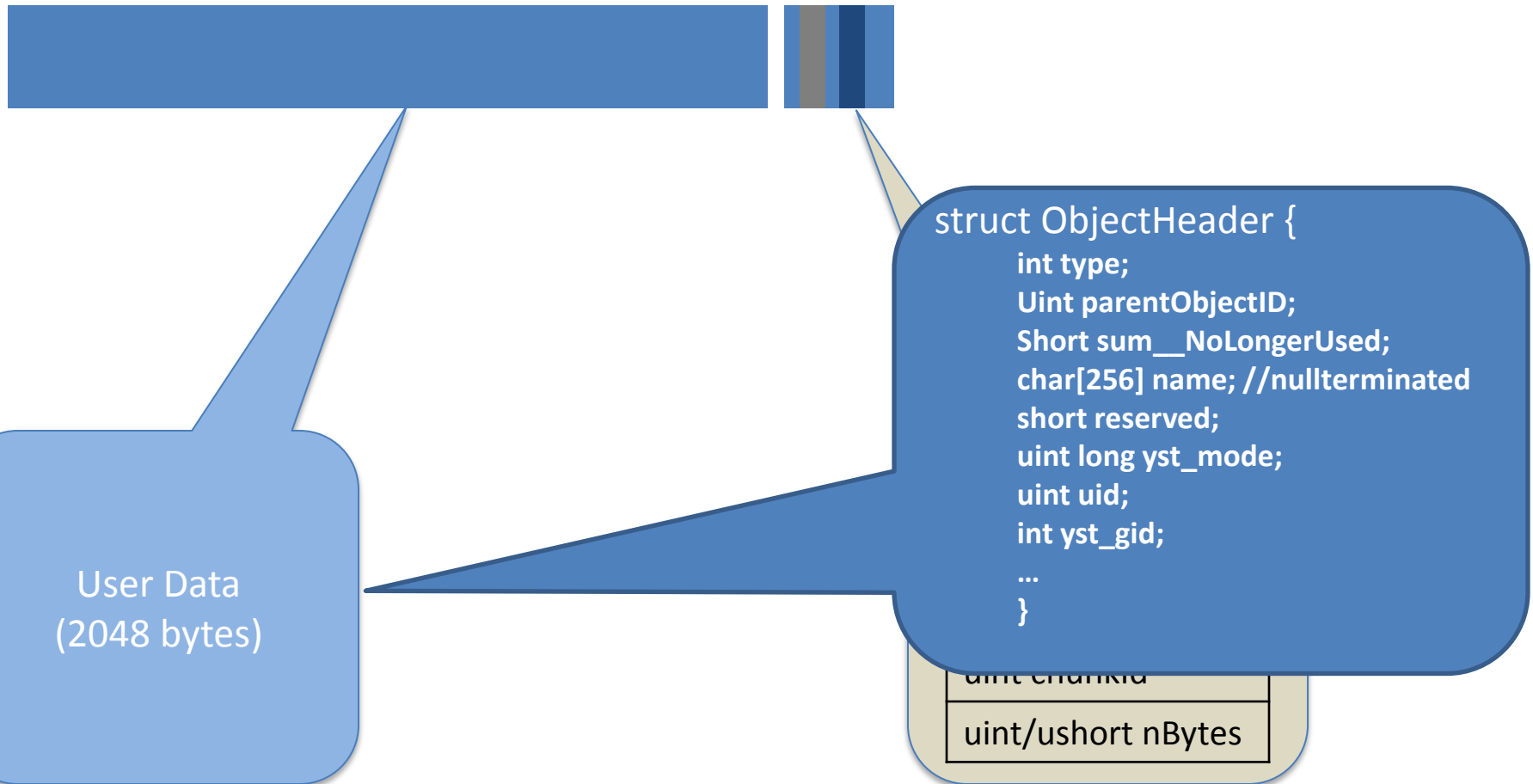
```
# cat /proc/mtd
dev:      size      erasesize  name
mtd0: 000a0000 00020000 "misc"
mtd1: 00500000 00020000 "recovery"
mtd2: 00280000 00020000 "boot"
mtd3: 0fa00000 00020000 "system"
mtd4: 02800000 00020000 "cache"
mtd5: 093a0000 00020000 "userdata"
```

YAFFS2 Volumes are distinguished by Object Header Striations

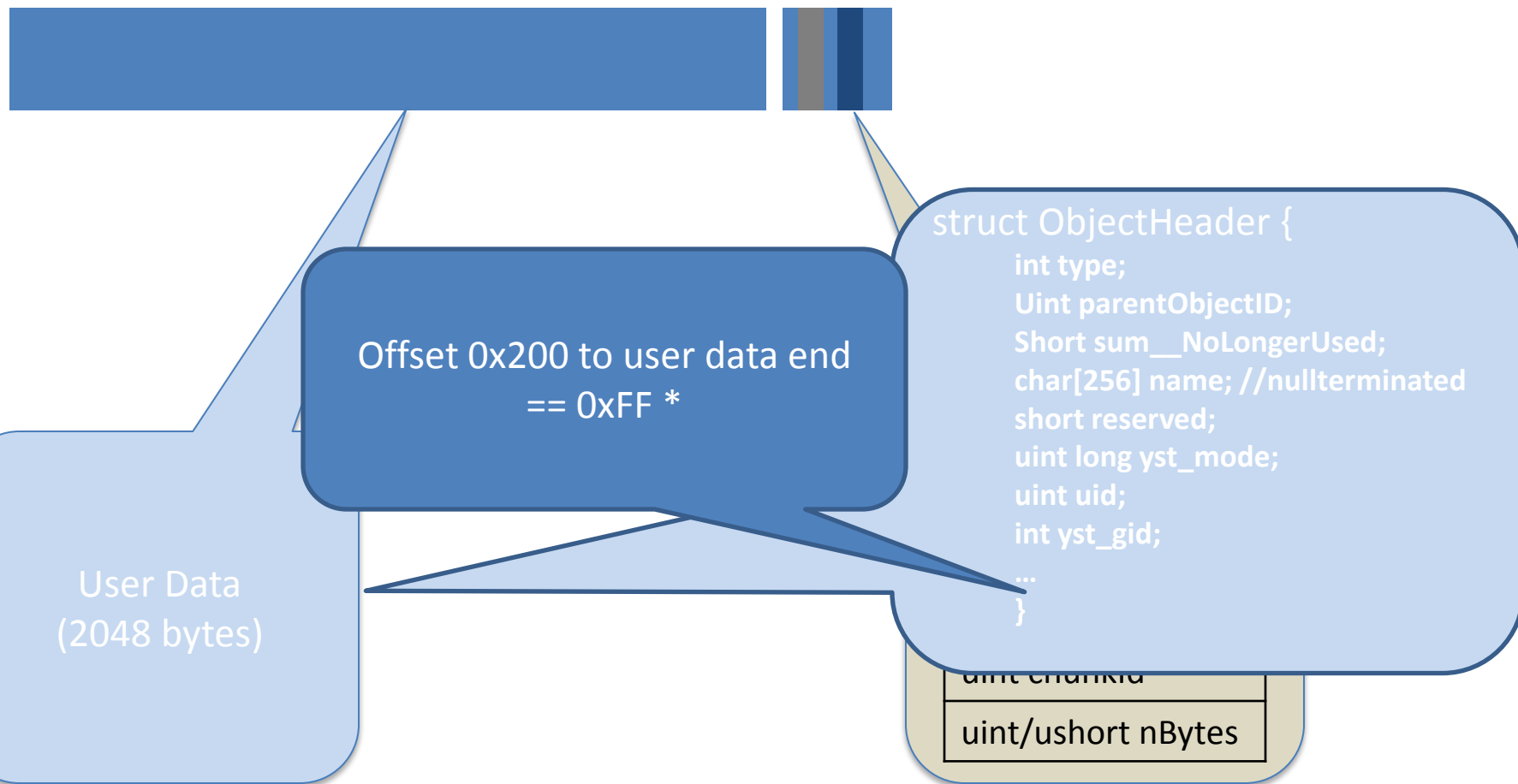


JTAG dump of
HTC Desire

YAFFS2 File (Object) metadata is stored in the user data area

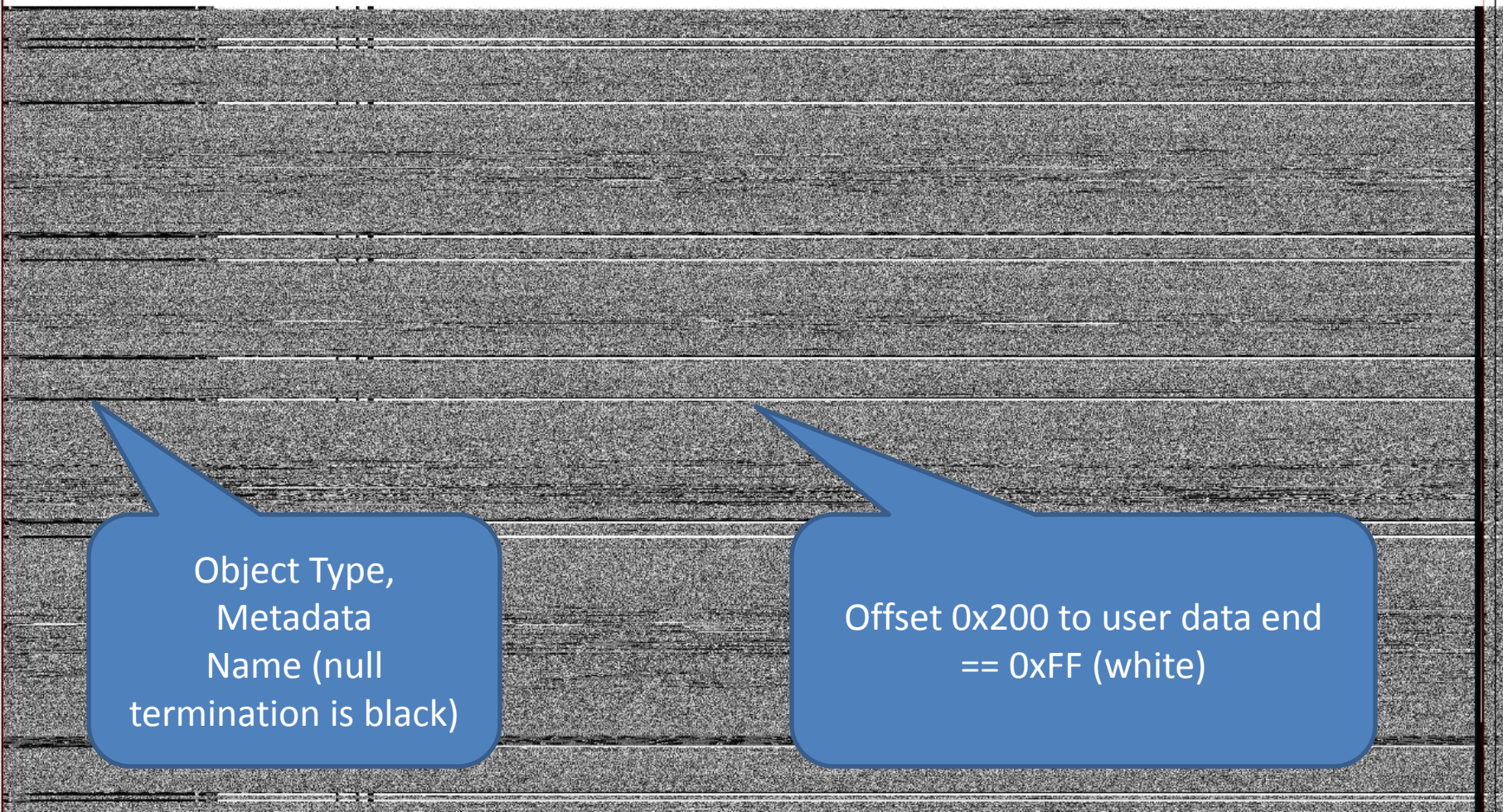


YAFFS2 File (Object) metadata is stored in the user data area



* Carving criteria identified by Pooters (2011) Yaffs2 Object Headers DFRWS

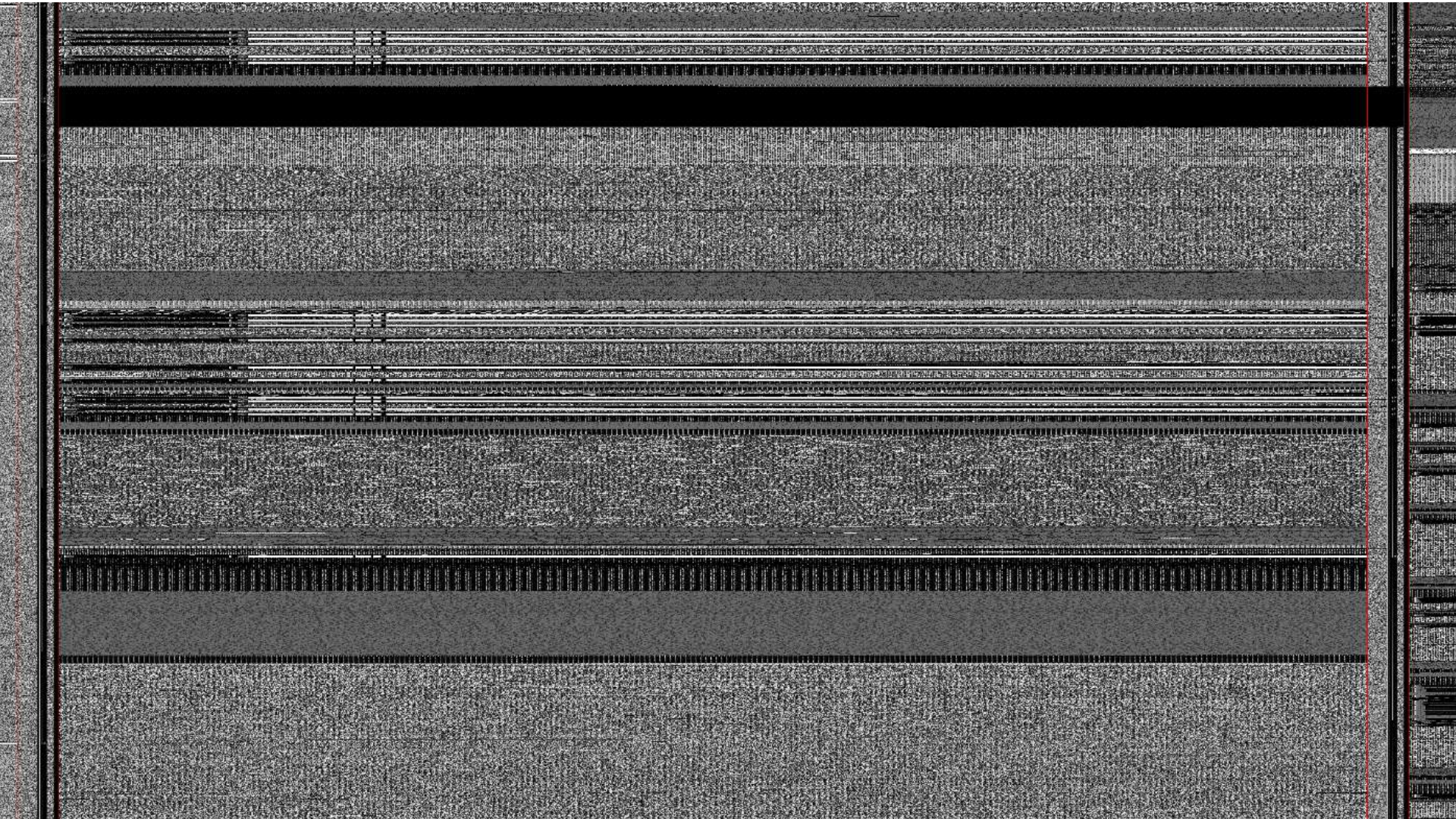
Object Header Striations Interpreted



Object Type,
Metadata
Name (null
termination is black)

Offset 0x200 to user data end
== 0xFF (white)

Theory indicates that Packed Tags and ECC should be in spare



YAFFS	2	Spare	(64	bytes)
consists of;				

bytes	
4	Chunk ID (20) (if 0 is a header (directory entry) if > 1 is data and position)
4	ObjectID (0 if unused)
2	nBytes, number of bytes used in the chunk, 0x 00 08 = 0x0800 = 2048 = full
4	Sequence number
3	ECC for tags
24	ECC for data
1	block status (damaged)
1	data status (dirty)

	00	01	02	03	04	05
00			Block ID			
10	-size		ECC			
20						

[illegible]

Figure 6. 'Extended Tag' structure of Droid flash memory

Source: Quick & Alzabbi (2011) Forensic analysis of the Android File System YAFFS2

Source: Bang et al (2011) DFRWS 2011 Forensic Challenge

YAFFS2 Basics: Simple file write



Object Header

ChunkID = 0

File Name = "a"

Sequence = 0x1001

Size = 0

Data block 1

ChunkID = 1

Address = 0x0

Sequence = 0x1001

Data block 2

ChunkID = 2

Address = 2048

Sequence = 0x1001

Object Header

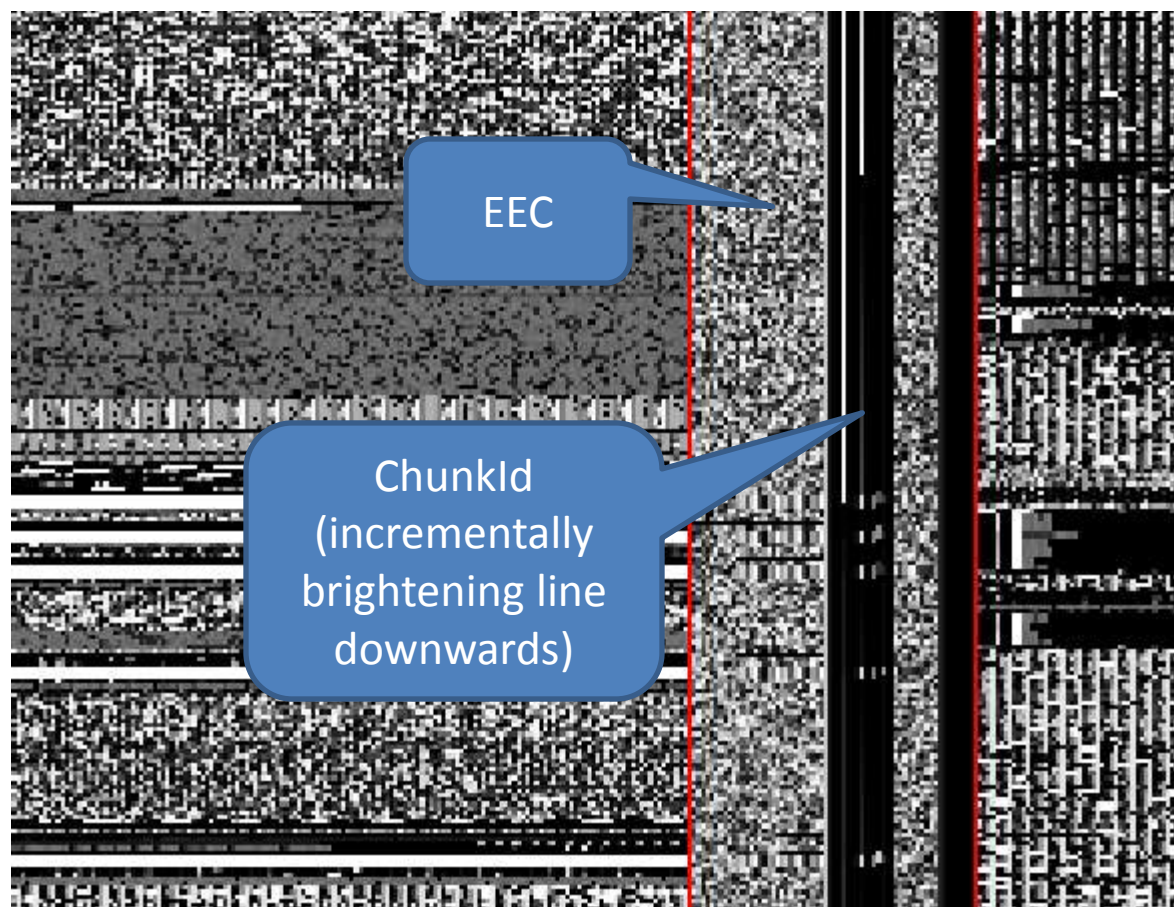
ChunkID = 0

File Name = "a"

Sequence = 0x1001

Size = 4096

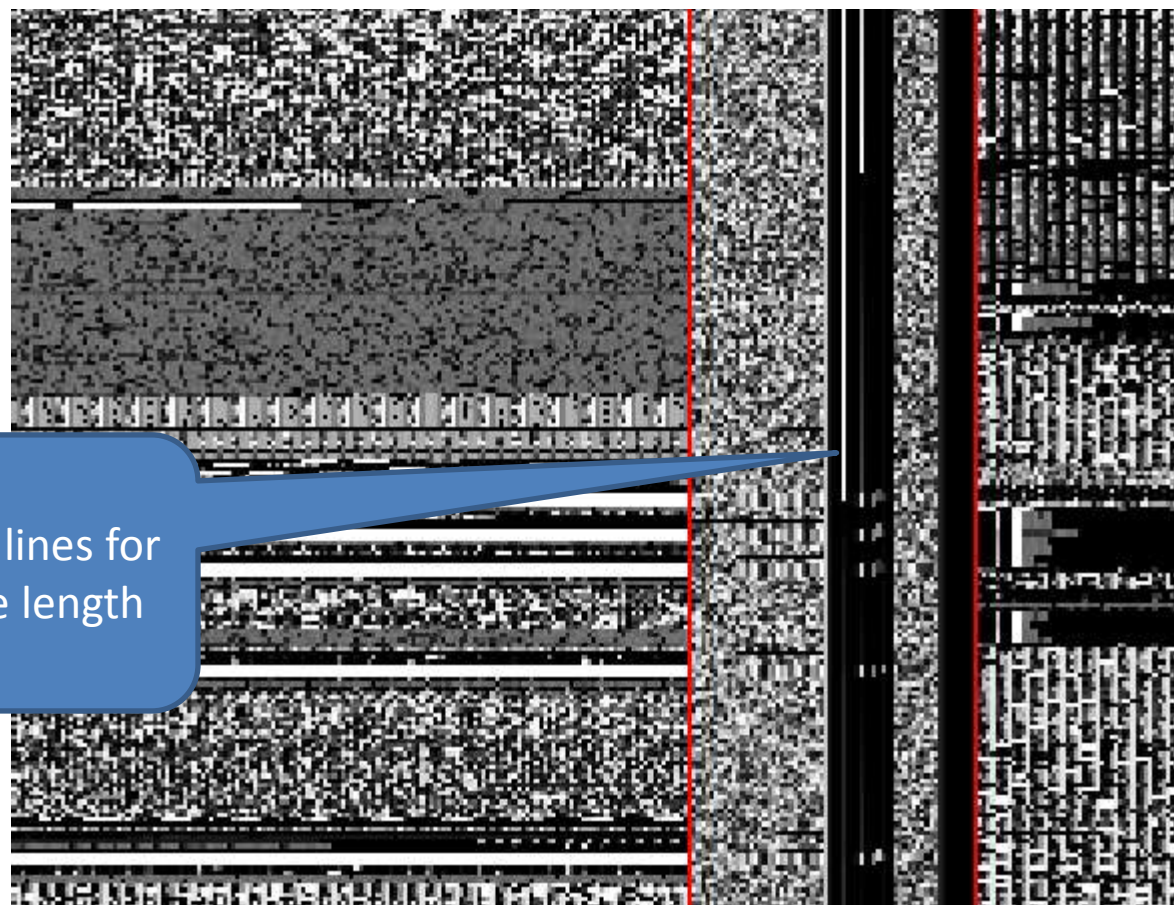
The ChunkID is distinguishable in sequentially written large files



Source:

© 2012 Schatz Forensic

The Sequence Number is Constant within the Erase Block



Constant lines for
blocksize length

Source:

© 2012 Schatz Forensic

A spare of 56 doesn't seem consistent with our current physical flash theory

```
# ./nanddump --bb=dumpbad -o -f ./mtd0.nanddump  
/dev/mtd/mtd0
```

ECC failed: 0

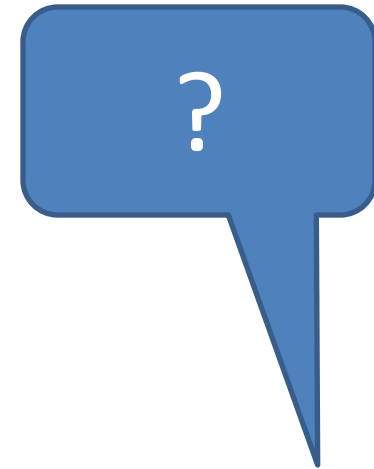
ECC corrected: 0

Number of bad blocks: 0

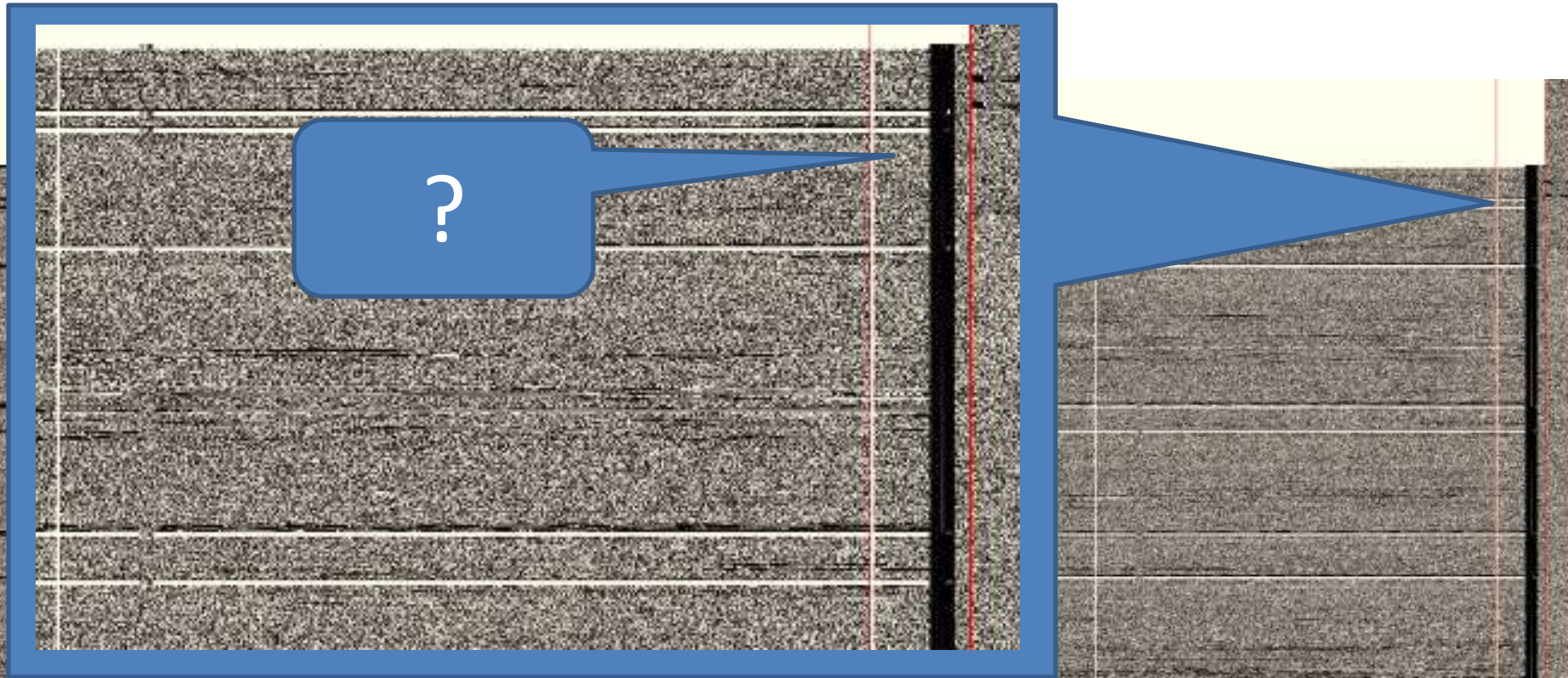
Number of bbt blocks: 0

Block size 131072, page size 2048, OOB size 56

Dumping data starting at 0x00000000 and ending at 0x000a0000...

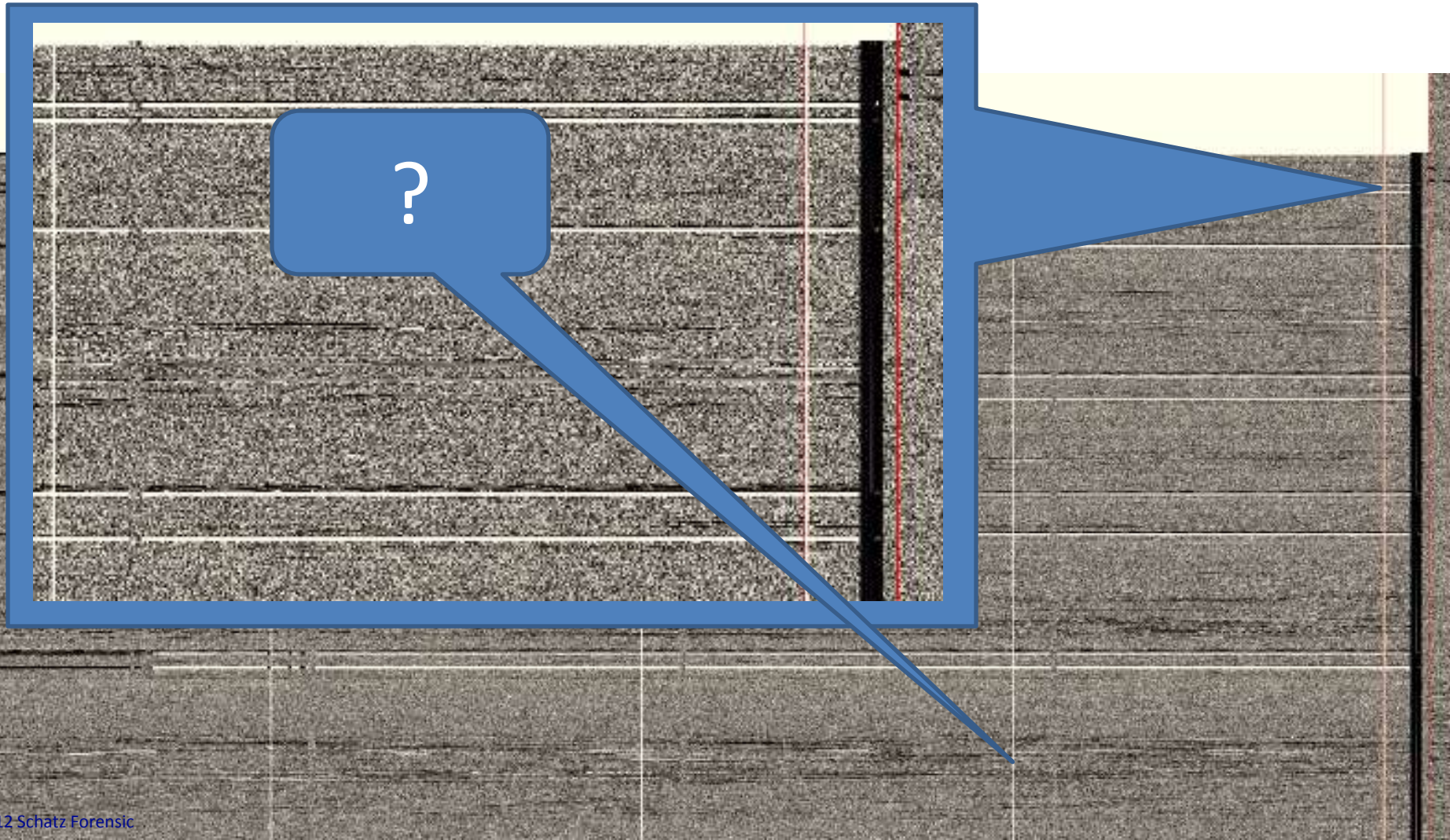


Why is the Object Header over filling the User Data area ?

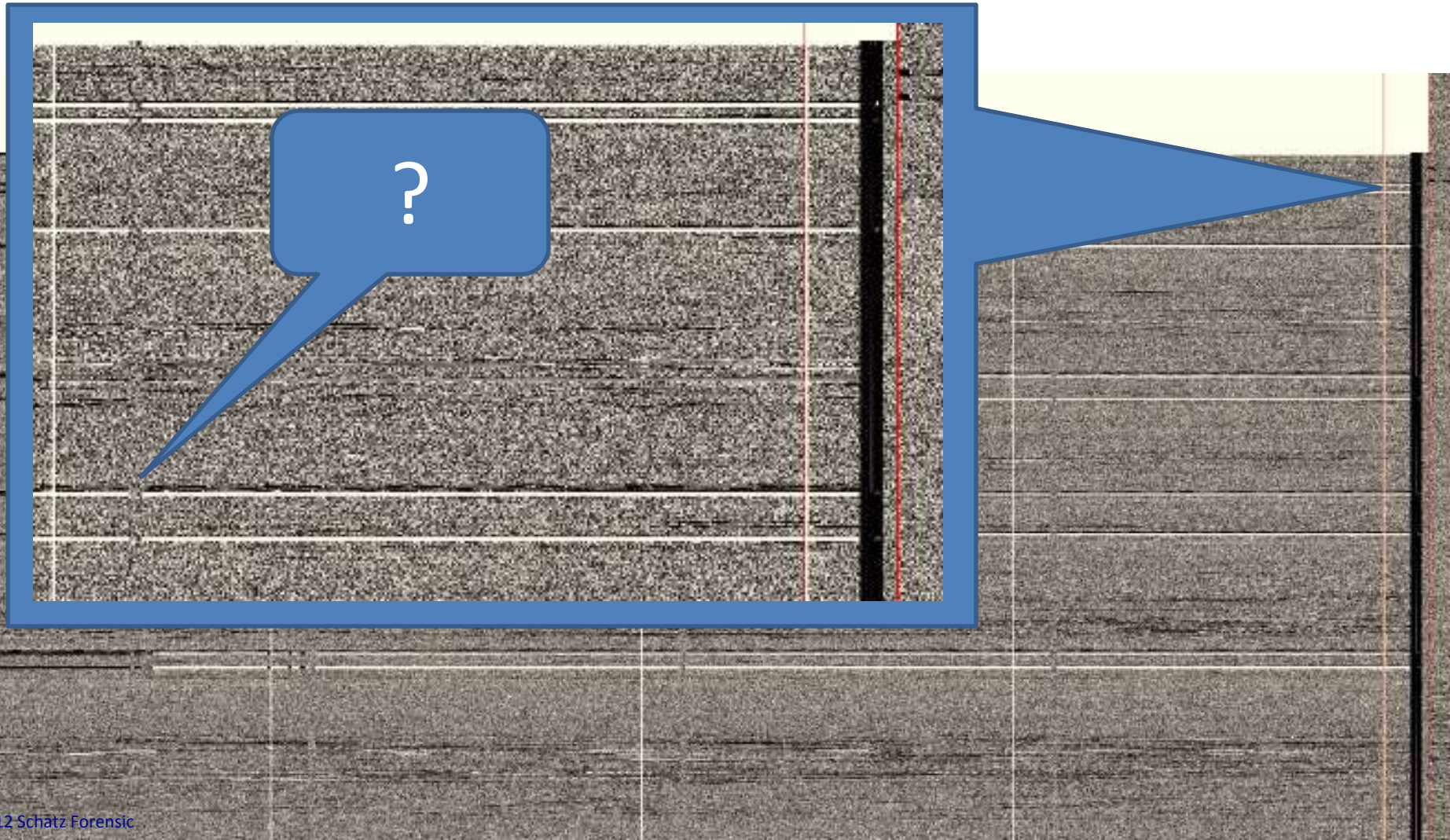


Chip off dump of KA100O015M-AJTT

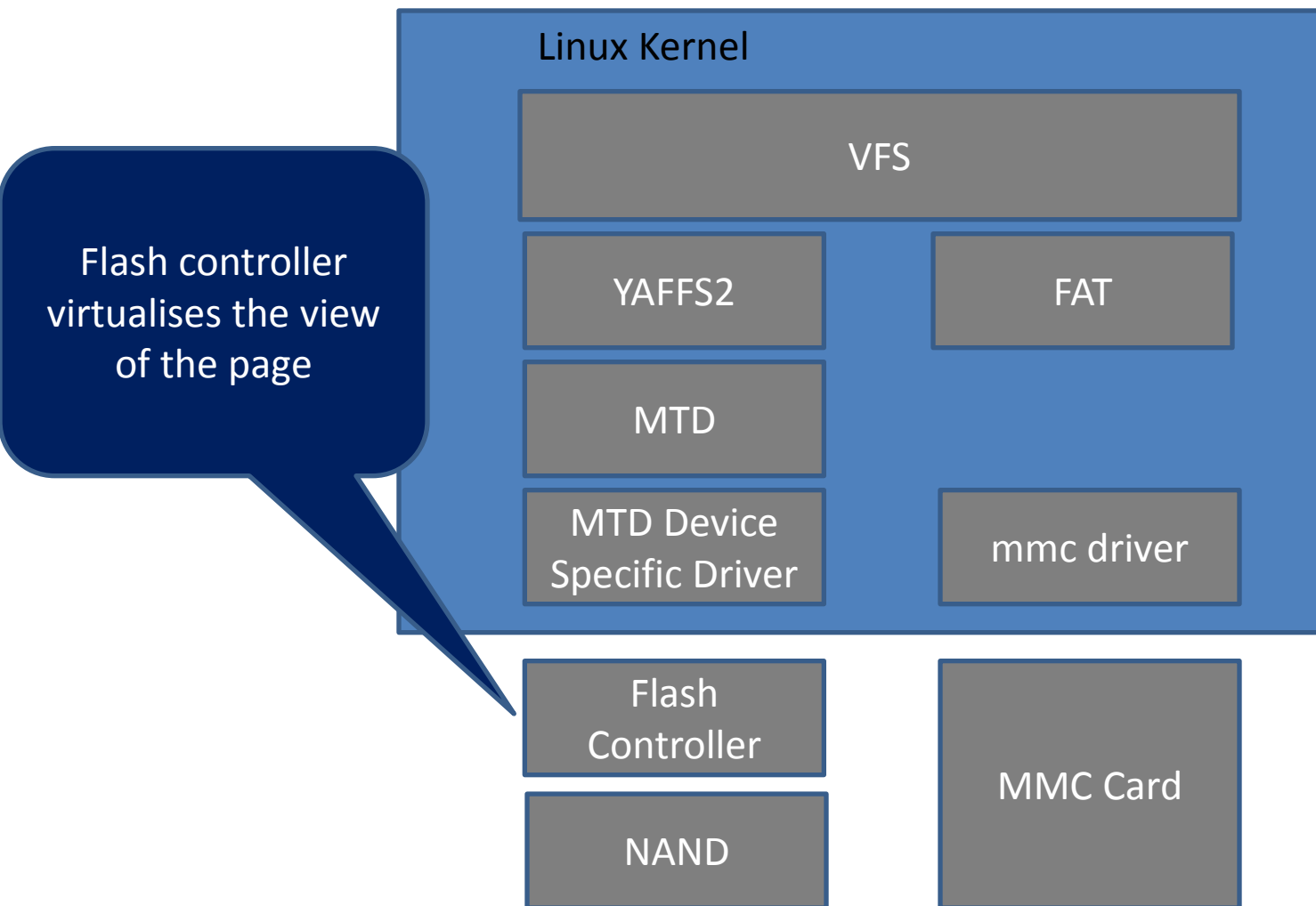
And what are these vertical lines?



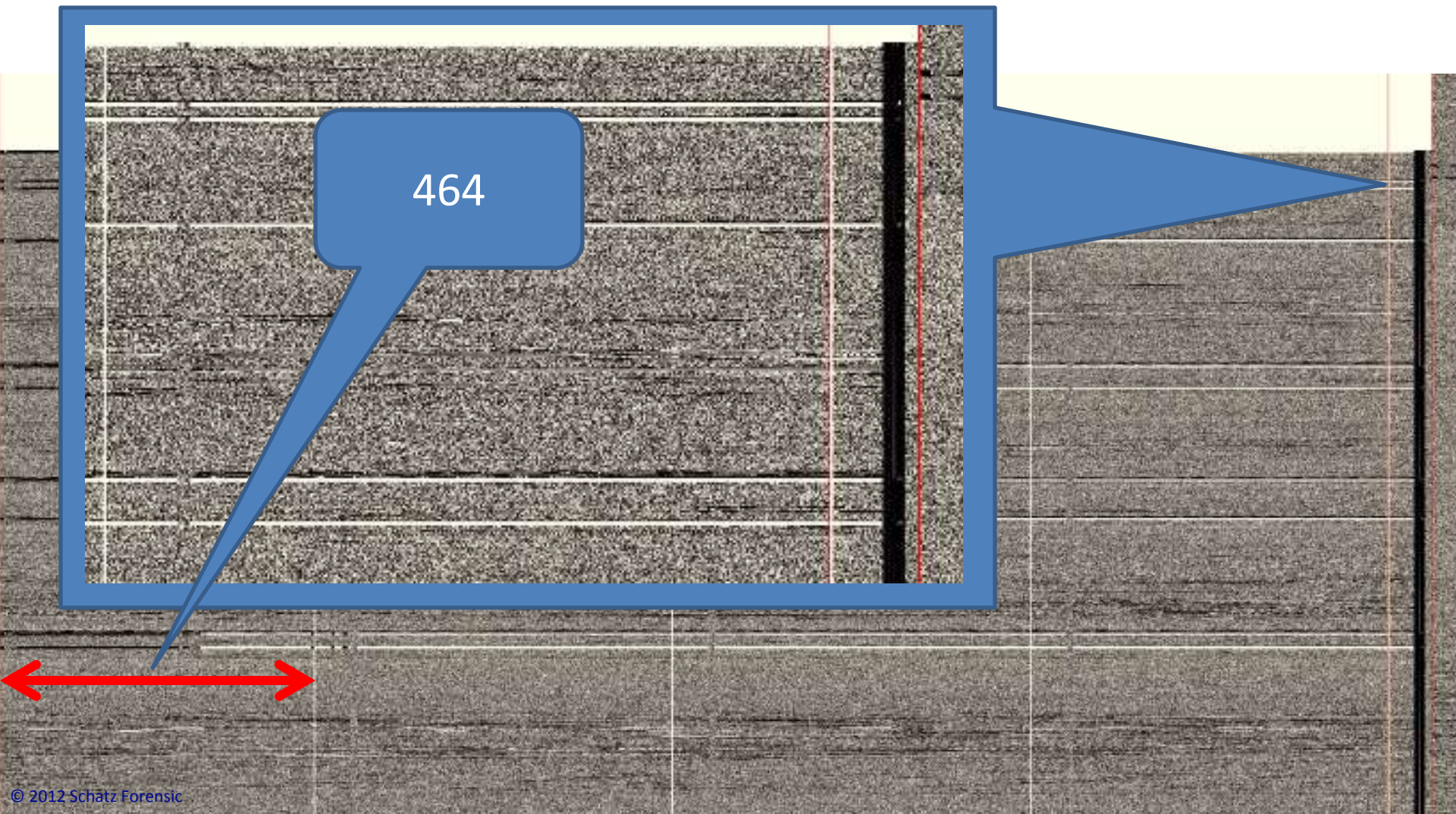
Is that EEC (note the high entropy) in
the user data area?



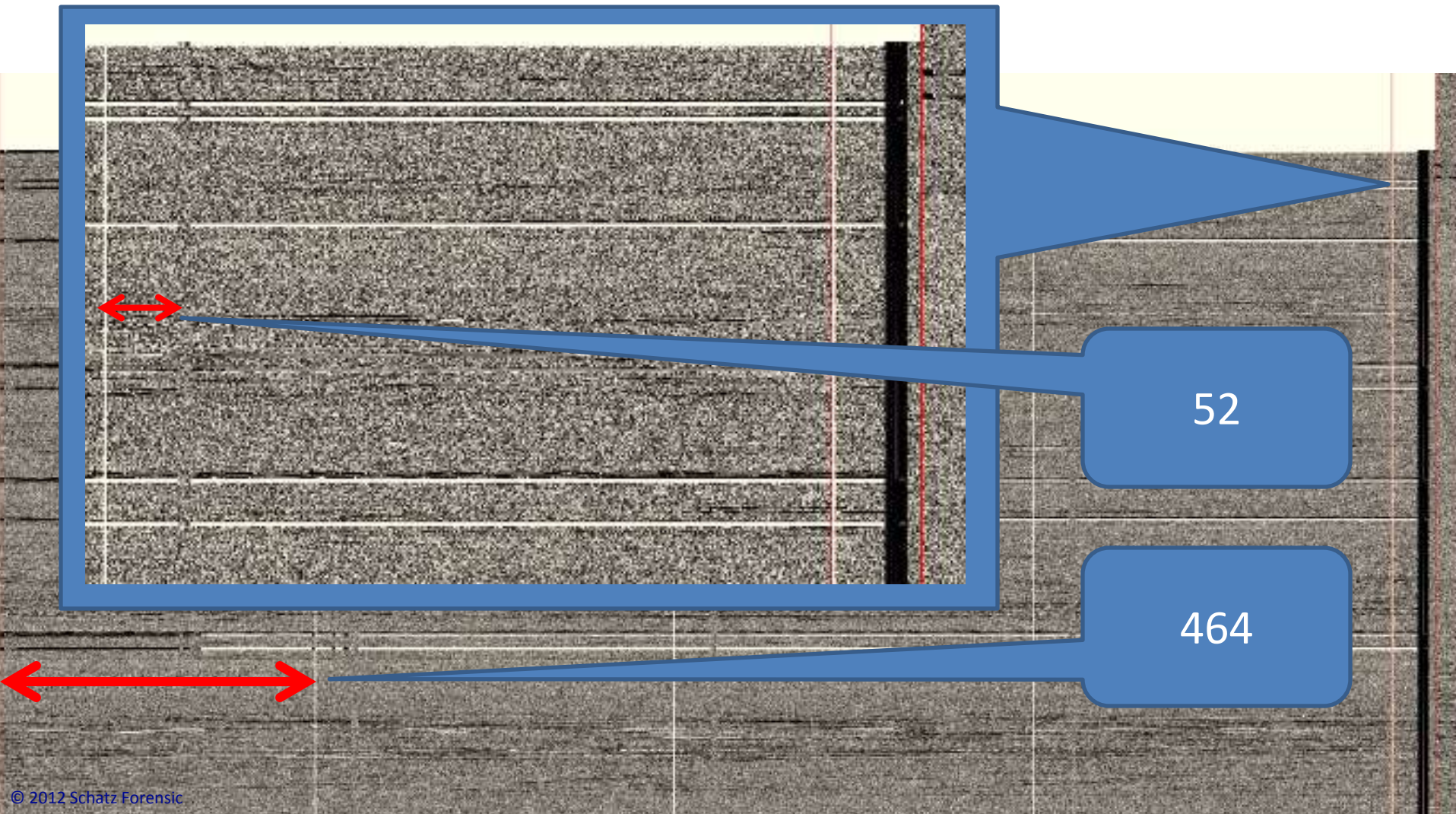
The Flash Controller is potentially the problem



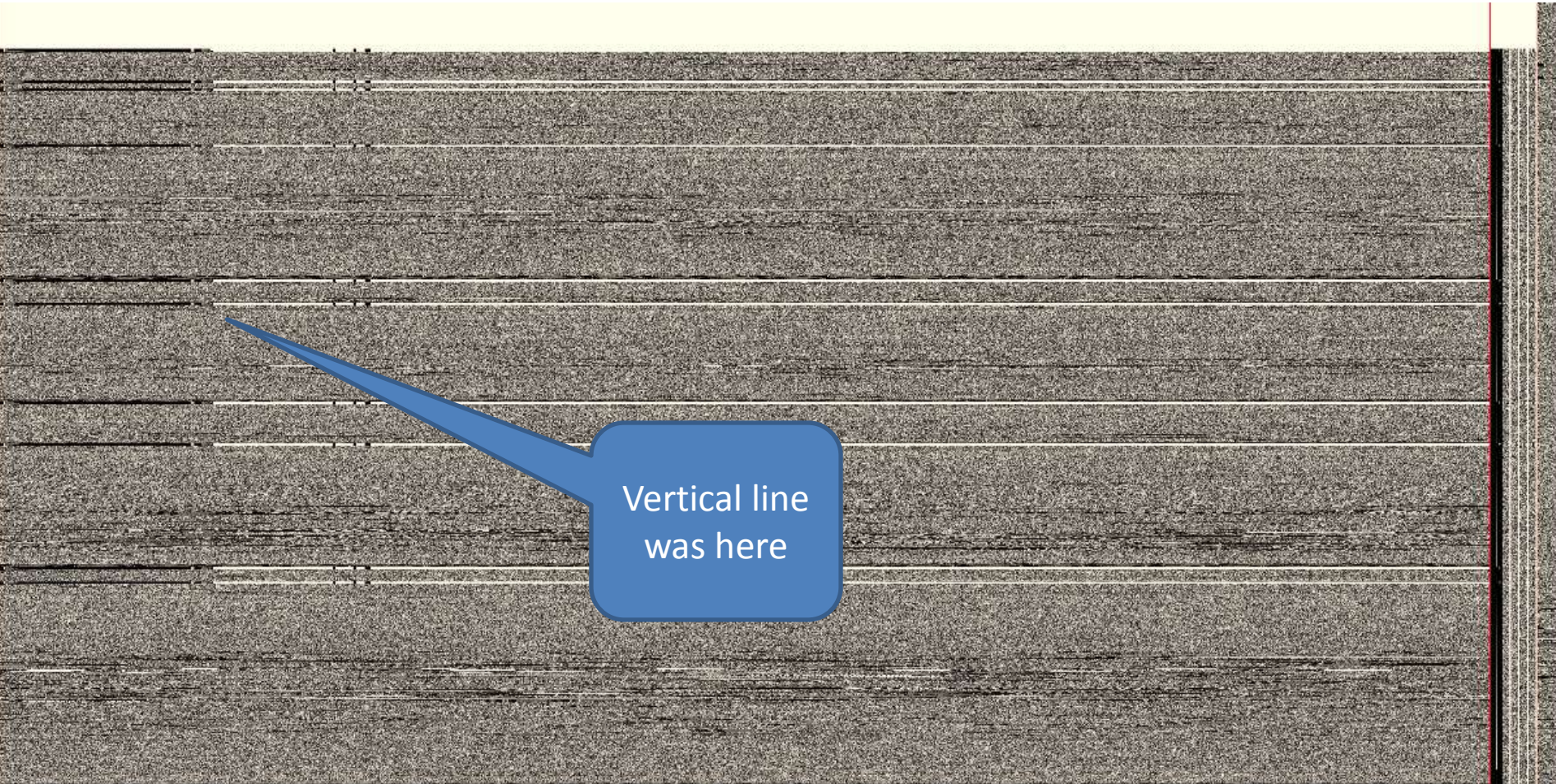
Why is the Object Header over filling the User Data area



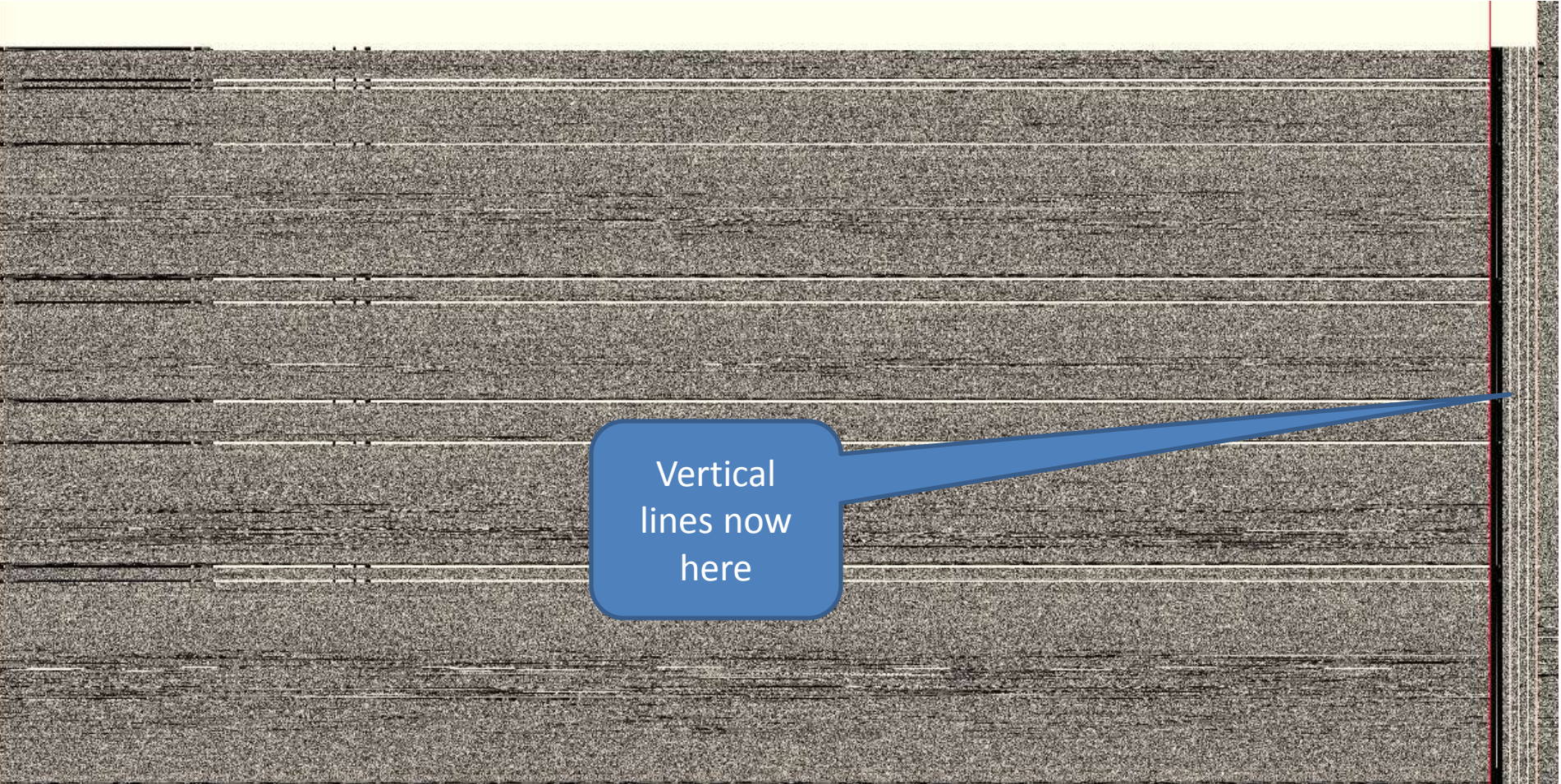
Why is the Object Header over filling the User Data area



Column relocation

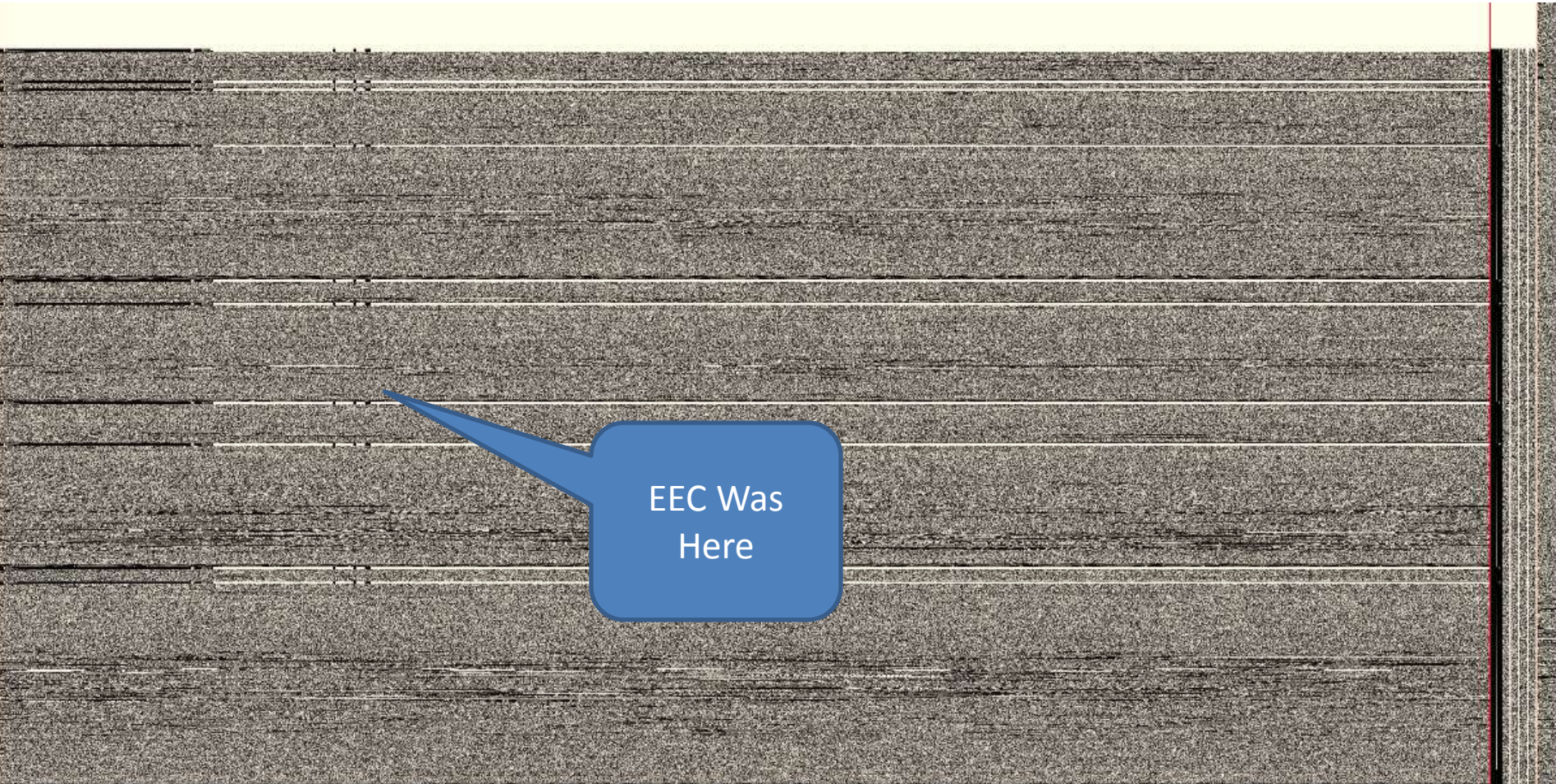


Column relocation



Vertical
lines now
here

Column relocation



Column relocation

An aerial photograph of a multi-lane highway interchange. A blue callout box with a pointer is overlaid on the image, indicating a specific location on the road.

EEC Now
Here

Column relocation



Everything
else is
shifted left



Analysis of the YAFFS2 Filesystem



Current freely available YAFFS2 implementations don't generally work with physical images

- Variable results with even pseudo physical images
- No support for prior object versions

YAFFS2 Sparse file creation



Object Header

ChunkID = 0

Sequence = 0x1001

File Name = "a"

Size = 0

Object Header

ChunkID = 0

Sequence = 0x1001

File Name = "a"

Size = 0 *sparse*

Object Header

ChunkID = 0

Sequence = 0x1001

File Name = "a"

Size = 0x1000

Object Header

ChunkID = 0

Sequence = 0x1001

File Name = "a"

Size = 0x1000

YAFFS2 Block Replace



Object Header

ChunkID = 0

Sequence = 0x1001

File Name = "a"

Size = 0

Data block 2

ChunkID = 1

Address = 0x0

Sequence = 0x1001

Expired

Data block 2

ChunkID = 1

Address = 0x0

Sequence = 0x1002

New data block

Object Header

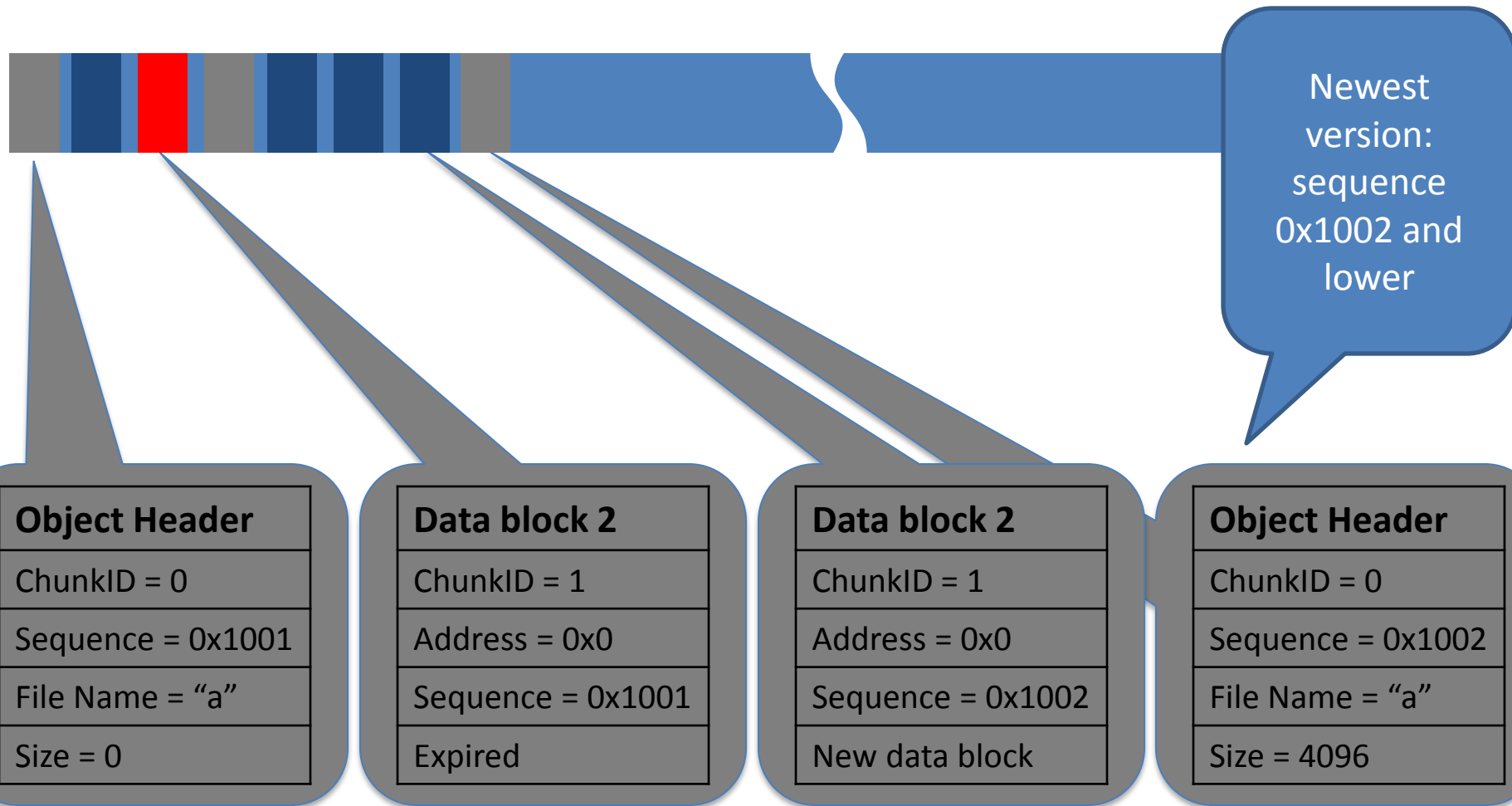
ChunkID = 0

Sequence = 0x1002

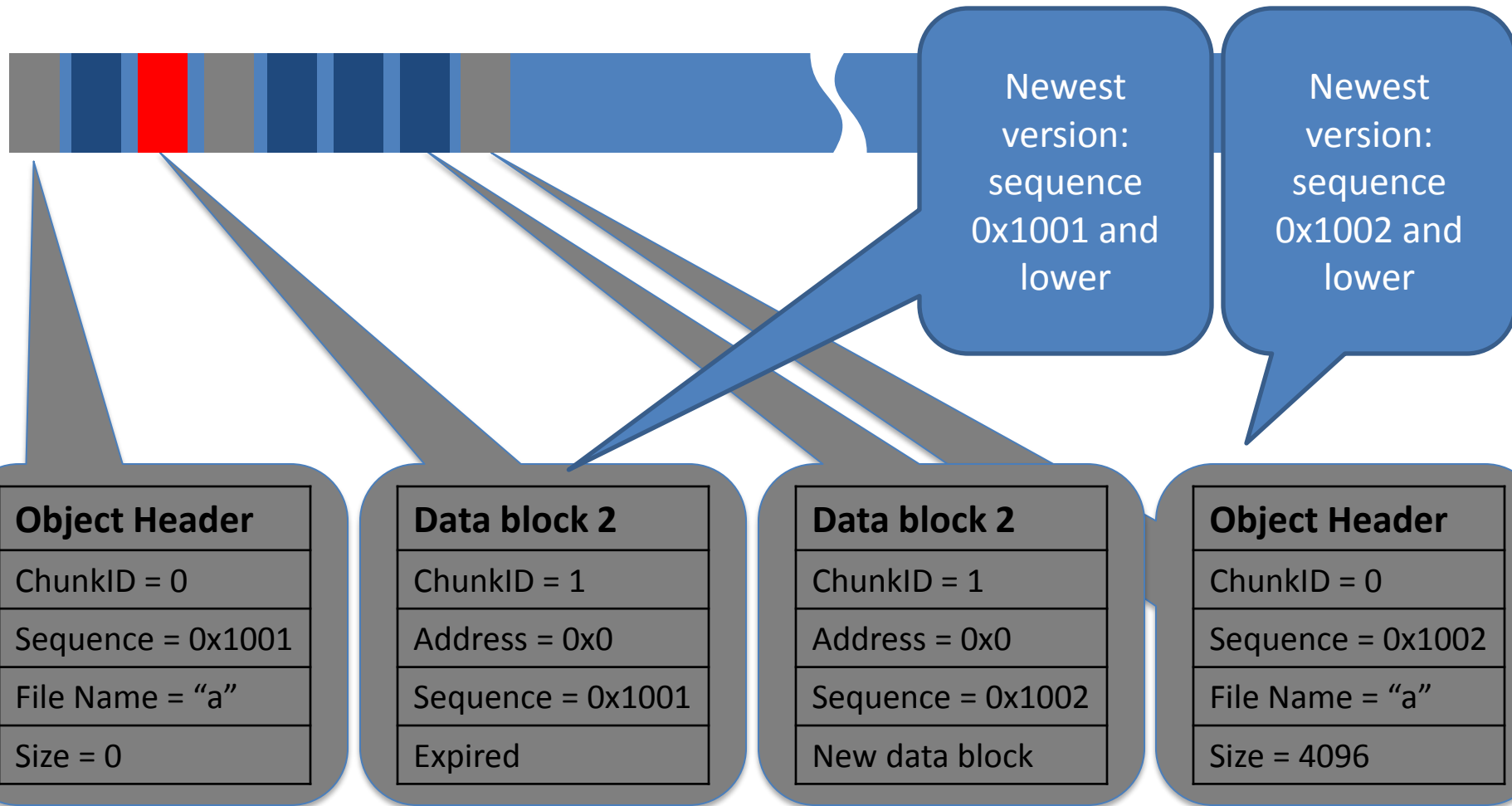
File Name = "a"

Size = 4096

YAFFS2 Version Recovery



YAFFS2 Version Recovery





Acquisition methodology

Acquisition Methodology

- JTAG or RAM Bootloader Acquisition
 - Recover PIN
- Live acquisition
 - Use PIN if necessary
 - Disable radios
 - Enable ADB
 - Exploit (you have validated it yes?)
 - Collect dmesg, /proc/mtd
 - Pseudo physical acquisition
 - Logical acquisition (for validation)



Conclusions



Contributions

- Byte plots assist in identifying structure in raw byte sequences
- Inconsistencies in prior research resolved in part
- Visual artefacts corresponding to structural elements identified
- A general acquisition methodology for JTAG based analysis proposed



Future Work

- Partitioning
- Automated normalisation
- Effects of heat on NAND integrity
- JTAG for Volatile Memory Analysis ?

- eMMC

Acknowledgements

- Andrew Hoog and co (Viaforensics)
 - For early and ongoing research in this area
- Tim Vidas
 - For YAFFS2 test VM
- DFRWS
 - For posing challenges that drive research



Dr Bradley Schatz | Forensic Computer Scientist
Director, Schatz Forensic
Adjunct Associate Professor, QUT

web: <http://schatzforensic.com.au/>
email: bradley@schatzforensic.com.au